



Setting up the development environment

Corso di Laurea in Informatica, Programmazione Distribuita
Delfina Malandrino, dmalandrino@unisa.it
<http://www.unisa.it/docenti/delfinamalandrino>

1

Organizzazione della lezione

2

- Setup dell'ambiente di sviluppo
 - NetBeans e GlassFish
 - Netbean 8, GlassFish v. 5.0
- Come caricheremo i progetti
- Primo esempio (injection in una servlet)
 - Computazione lato Server HTTP
- Conclusioni

2

Installazione di NetBeans

3

- Download NetBeans 8.2
 - Java EE (non scaricare GlassFish Server)

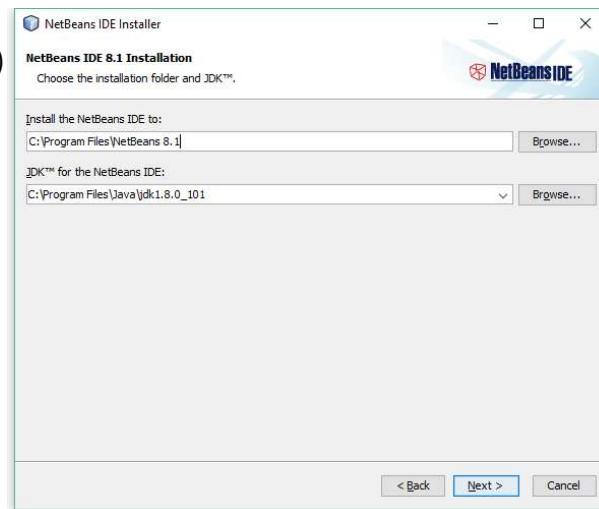


3

Installazione di NetBeans

4

- Download NetBeans 8.2
 - Java EE (non scaricare GlassFish Server)

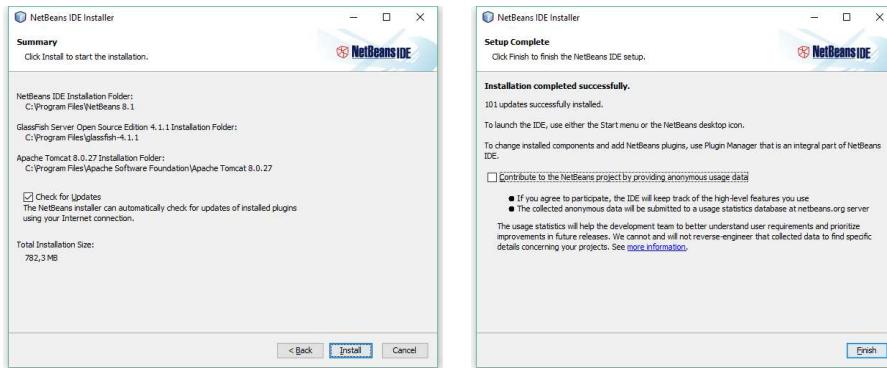


4

Installazione di NetBeans

5

- Download NetBeans 8.2
 - Java EE (non scaricare GlassFish Server)



5

GlassFish: application server di riferimento

6

- Reference Implementation
- Non solo il server “di default” (scaricato con Java EE)
- ... ma anche un server che può essere usato per applicazioni industriali
- Origini che vanno al server Tomcat, nel 2005 (primo shipping nel 2006)
- Comunità viva e vitale, con documentazione di buona qualità
- Versione “community” (=free) e “supported” (con tool aggiuntivi di monitoraggio, ma con lo stesso core)

6

Architettura di GlassFish

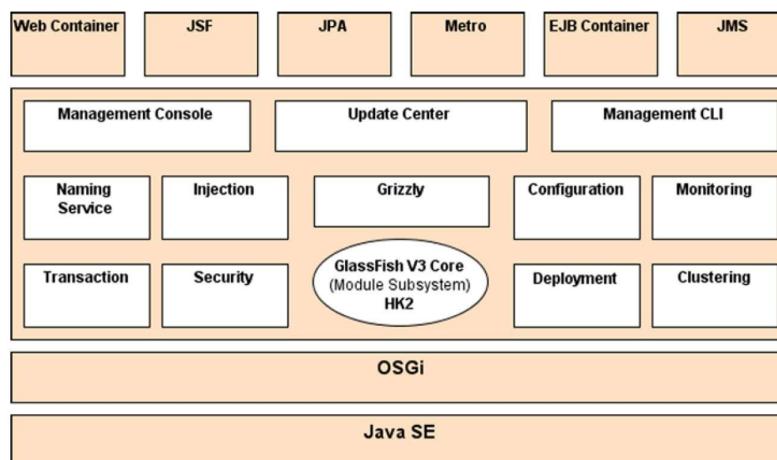
7

- Come programmati, abbiamo bisogno di capire la architettura a “grana grossa” del server (dettagli non strettamente influenti sulle funzionalità fornite)
- Costruito su un kernel modulare basato su standard OSGi
- OSGi è uno standard (commerciale Ericsson, IBM, Oracle) per la gestione dinamica di componenti nei sistemi, e la loro scoperta e utilizzo senza reboot
 - componenti che riconoscono l’aggiunta/rimozione di altre componenti a run-time
- Diversi sistemi sono costruiti su standard OSGi (Apache Felix implementazione free, Equinox (Eclipse))
- Caratteristiche: leggerezza (utile per testing in-site)
- Installazione standard: 8080 server, 4848 console di admin

7

Architettura

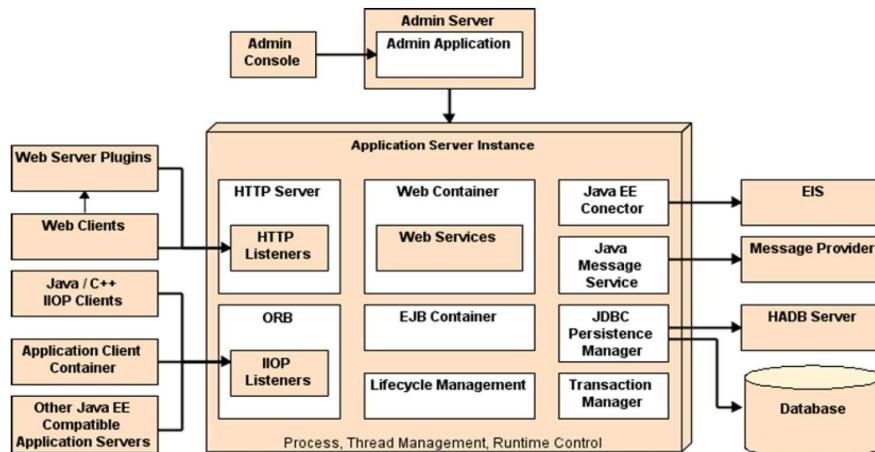
8



8

Funzionalità

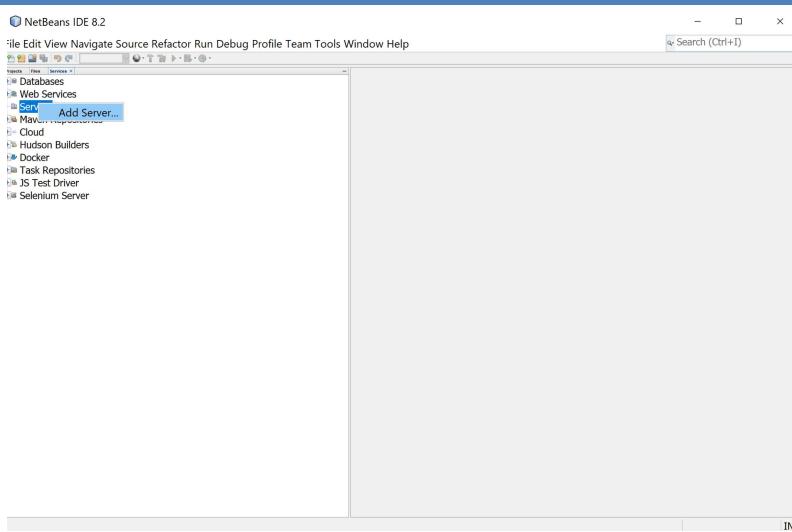
9



9

Aggiungiamo Glassfish

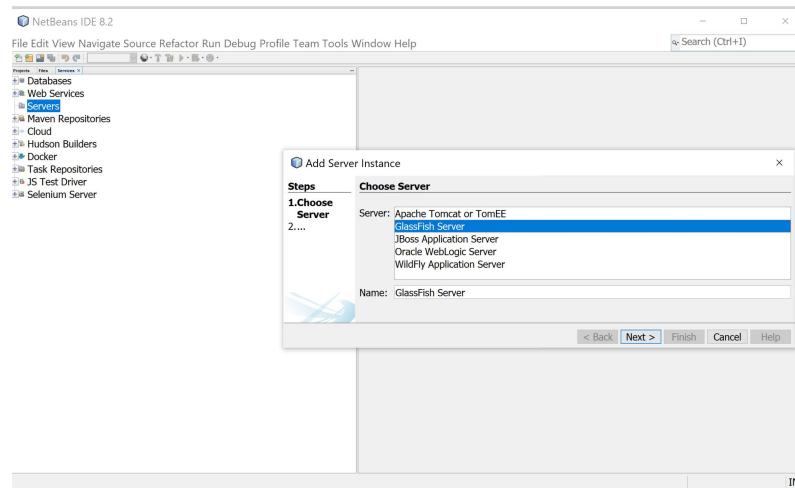
10



10

Aggiungiamo Glassfish

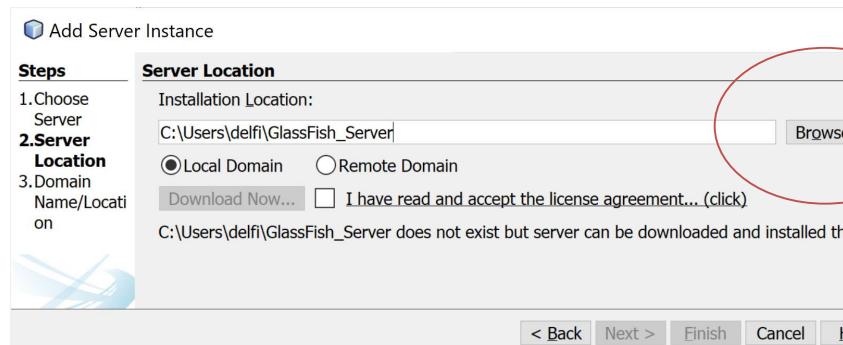
11



11

Aggiungiamo Glassfish

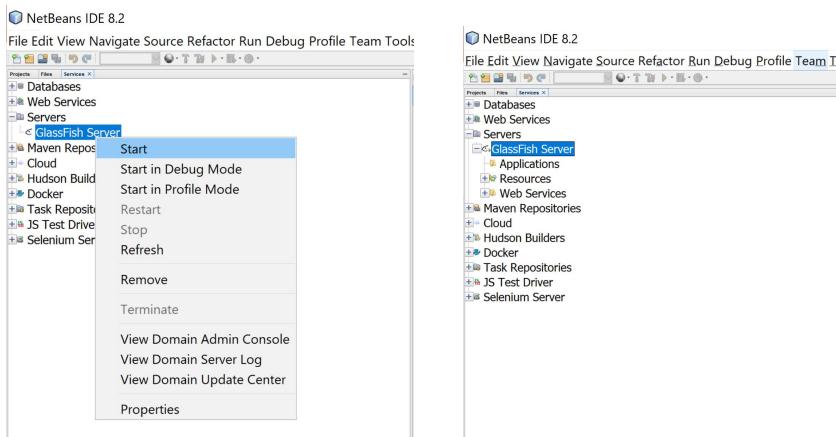
12



12

Aggiungiamo Glassfish

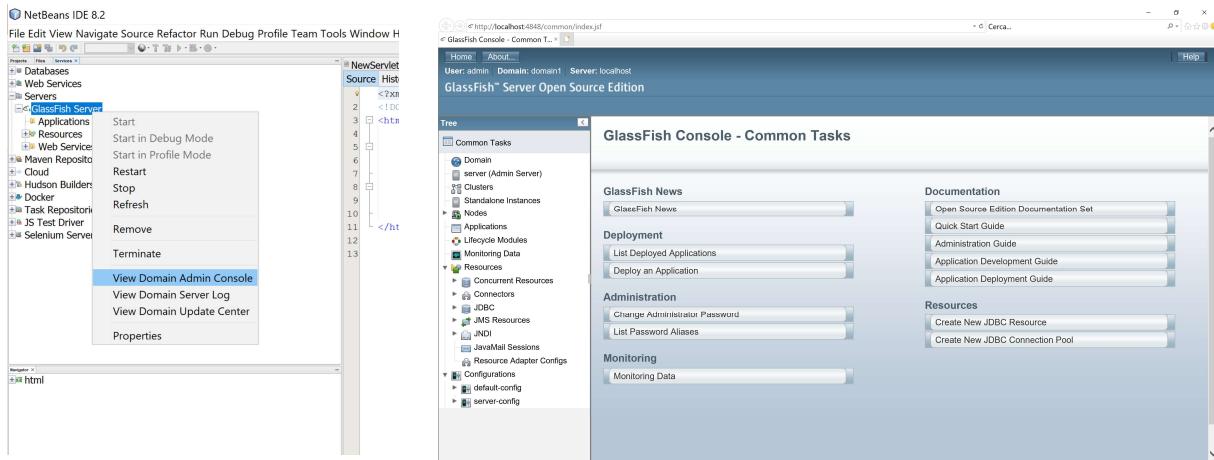
13



13

Aggiungiamo Glassfish

14



14

Organizzazione della lezione

15

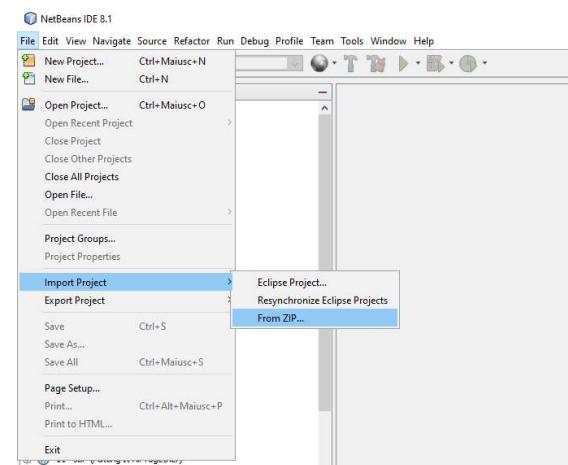
- Setup dell'ambiente di sviluppo
 - NetBeans e GlassFish
- Come caricheremo i progetti
- Primo esempio (injection in una servlet)
 - Computazione lato Server HTTP
- Conclusioni

15

Importiamo i progetti

16

- Caricamento progetti



16

Organizzazione della lezione

17

- Setup dell'ambiente di sviluppo
 - NetBeans e GlassFish
- Come caricheremo i porgetti
- Primo esempio (injection in una servlet)
 - Computazione lato Server HTTP
- Conclusioni

17

Primo esempio

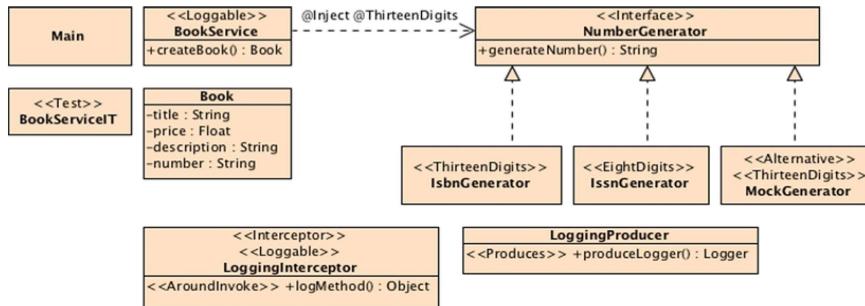
18

- Primo esempio di una applicazione Java Enterprise
- Focus sulla struttura e l'utilizzo di CDI
- Struttura dell'esempio: una servlet che fa injection di una istanza di BookService
 - su cui fa una semplice richiesta di creazione di un libro

18

La struttura dell'esempio

19



Putting It All Together

19

Writing the Book and BookService classes

20

Listing 2-40

```

public class Book {
    private String title;
    private Float price;
    private String description;
    private String number;
    //Constructors, getters, setters
}
  
```

20

Writing the Book and BookService classes

21

Listing 2-41

```
@Loggable
public class BookService {

    @Inject @ThirteenDigits
    private NumberGenerator numberGenerator;

    public Book createBook(String title, Float price, String description)
    {
        Book book = new Book(title, price, description);
        book.setNumber(numberGenerator.generateNumber());
        return book;
    }
}
```

1. Il BookService è annotato con l'interceptor binding `@Loggable` (logging di entry and exit dal metodo)
2. Il BookService usa injection (`@Inject`) e un qualificatore (`@ThirteenDigits`) per invocare il metodo `generateNumber` di `ISBNGenerator` per settare il numero ISBN del libro

21

Writing the NumberGenerator classes

22

Listing 2-42

```
public interface NumberGenerator {
    String generateNumber();
}
```

1. Il BookService dipende dall'interfaccia `NumberGenerator`. Questa interfaccia ha un metodo che genera e restituisce un book number

22

Writing the NumberGenerator classes

23

Listing 2-43

```
@ThirteenDigits
public class IsbnGenerator implements NumberGenerator {

    @Inject
    private Logger logger;

    @Loggable
    public String generateNumber() {
        String isbn ="13-84356-"+ Math.abs(new Random().nextInt());
        logger.info("Generated ISBN:"+ isbn);
        return isbn;
    }
}

1. L'interfaccia è implementata dalle classi IsbnGenerator, IssnGenerator, e MockGenerator
2. La classe IsbnGenerator ha un qualificatore (@ThirteenDigits). Questo per informare CDI che il numero
generato è di 13 digits.
3. La classe IsbnGenerator usa injection per ottenere un java.util.logging.Logger ed un interceptor
binding @Loggable per loggare method entry ed exit
```

23

Writing the NumberGenerator classes

24

Listing 2-44

```
@EightDigits
public class IssnGenerator implements NumberGenerator {

    @Inject
    private Logger logger;

    @Loggable
    public String generateNumber() {
        String issn ="8-"+ Math.abs(new Random().nextInt());
        logger.info("Generated ISSN:"+ issn);
        return issn;
    }
}
```

24

Writing the Qualifiers

25

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface ThirteenDigits { }
```

Listing 2-46

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD})
public @interface EightDigits { }
```

Listing 2-47

1. Poiché esistono diverse implementazioni di `NumberGenerator`, CDI ha necessità di qualificare ogni bean ed ogni injection point per evitare injection ambigue.
2. Usa due qualificatori `ThirteenDigits` (Listing 2-46) e `EightDigits` (Listing 2-47) annotati con `javax.inject.Qualifier` e non ci sono membri

25

Writing the Logger

26

```
public class LoggingProducer {
    @Produces
    public Logger produceLogger(InjectionPoint injectionPoint) {
        return Logger.getLogger(injectionPoint.getMember().
            getDeclaringClass().getName());
    }
}
```

Listing 2-48

1. Il nostro esempio usa il logging in diversi modi. Le implementazioni di `NumberGenerator` usano injection per ottenere un `java.util.logging.Logger` e scrivere logs
2. Poiché Logger appartiene alla JDK, non può essere iniettato di default (rt.jar file non ha un beans.xml file), bisogna pertanto produrlo
3. La classe `LoggingProducer` ha un metodo `producer (produceLogger)` annotato con `@Produces` che creerà e restituirà un Logger parametrizzato con un injection point class name

26

Writing the Logger

27

```
@Interceptor
@Loggable
public class LoggingInterceptor {

    @Inject
    private Logger logger;

    @AroundInvoke
    public Object logMethod(InvocationContext ic) throws Exception {
        logger.entering(ic.getTarget().getClass().getName(),
                        ic.getMethod().getName());
        try{
            return ic.proceed();
        }finally{
            logger.exiting(ic.getTarget().getClass().getName(),
                           ic.getMethod().getName());
        }
    }
}
```

Listing 2-49

1. The `LoggingInterceptor` usa il Logger prodotto per loggare entering ed exiting dei metodi
2. Poiché il logging può essere trattato come un cross-cutting concern, può essere esternalizzato come un interceptor (`@AroundInvoke` sul `logMethod`)
3. Il `LoggingInterceptor` definisce il `@Loggable` interceptor binding e può essere usato in ogni bean (e.g., `BookService` in Listing 2-41)

27

Writing the Logger

28

```
@InterceptorBinding
@Target({METHOD, TYPE})
@Retention(RUNTIME)
public @interface Loggable {
}
```

Listing 2-50

28

Writing the Main Class

29

```
public class Main {
    public static void main(String[] args) {
        Weld weld = new Weld();
        WeldContainer container = weld.initialize();

        BookService bookService = container.instance().
            select(BookService.class).get();

        Book book = bookService.createBook("H2G2", 12.5f,
            "GeekySciFiBook");

        System.out.println(book);

        weld.shutdown();
    }
}
```

1. Creazione del Container (in Java SE!)
2. Inizializzazione
3. Restituisce una istanza costruita e "iniettata" di BookService
4. Invochiamo un metodo
5. Stampiamo il libro creato
6. Chiudiamo il server

29

Writing the Main Class

30

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                           http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       version="1.1" bean-discovery-mode="all">
</beans>
```

30

Computazione lato Server HTTP

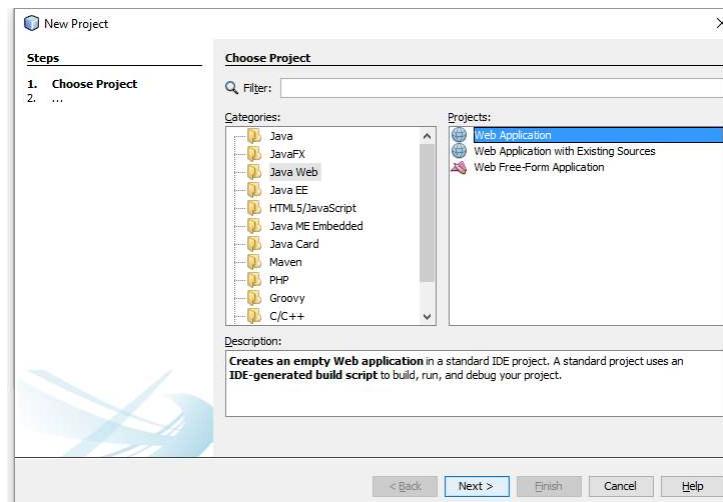
31

- Componenti di una Web Application che rispondono ad una richiesta HTTP
- Ricevono la richiesta, con tutti i parametri del protocollo HTTP
 - header, metodo (GET, POST, . . .), parametri passati, etc.
- Costruiscono una risposta in HTML che viene restituita al client
- Modello di computazione “antico” (un tempo si usavano le applicazioni esterne: Common Gateway Interface)
- Possibile inserirle in diversi ambienti server, da quelli più semplici (semplici Web container) ai veri e propri application server (GlassFish, JBoss, etc.)
- Vedremo un esempio di una servlet costruita con NetBeans per accedere ad una risorsa iniettata

31

Web Application in NetBeans

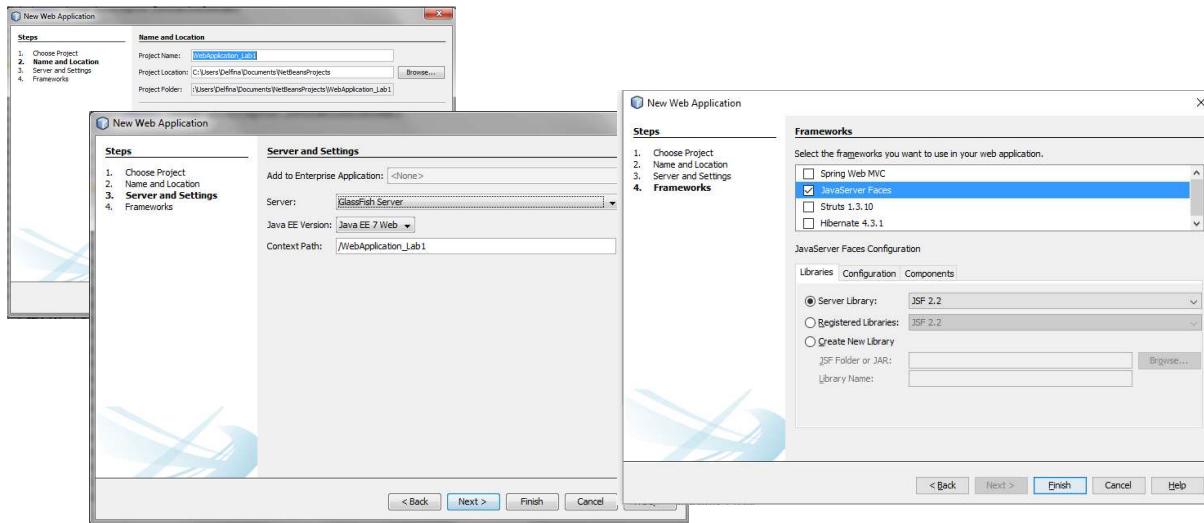
37



32

Web Application in NetBeans

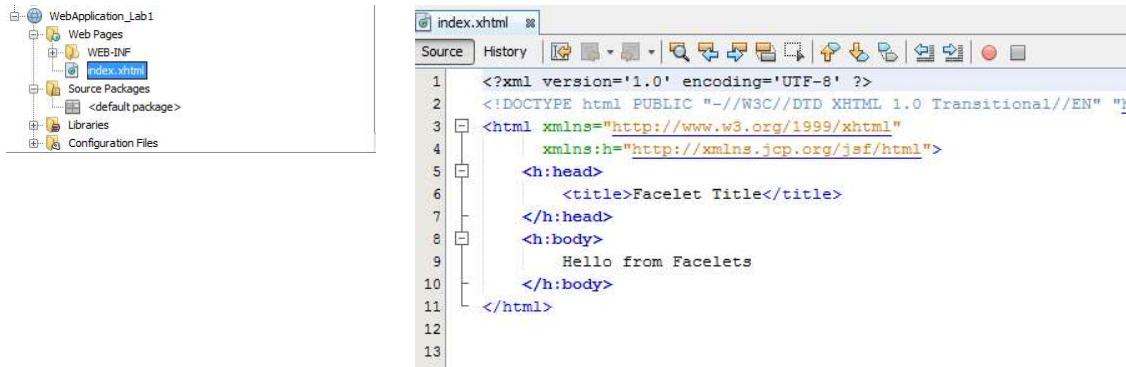
38



33

Web Application in NetBeans

39



34

Web Application in NetBeans: Build

35

□ Build & Deploy

The screenshot shows the NetBeans IDE interface. The top part is a Java code editor with the following code:

```
<%@page version="1.0" encoding="UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jst/faces">
<head>
    <title>Facelet Title</title>
</head>
<body>
    Hello from Facelets
</body>
</html>
```

The bottom part is a terminal window titled "Output: WebApplication_Labt [dist] # Test Results". It displays the build process:

```
Copying 1 file to C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\build\web\META-INF
Copying 2 files to C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\build\web\lib\jboss-inclusion-in-archives.jar
lib\jboss-inclusion-in-archives.jar
Created dir: C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\build\empty
Created dir: C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\build\generated-sources\ap-source\output
compile
compile-all-sources
compile-all-sources
Created dir: C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\dist
Building jar: C:\Users\Defina\Documents\NetBeansProjects\WebApplication_Labt\dist\WebApplication_Labt.war
do-dist
dist
BUILD SUCCESSFUL (total time: 0 seconds)
```

35

Web Application in NetBeans: esecuzione

36

Output



36

Cosa vogliamo fare

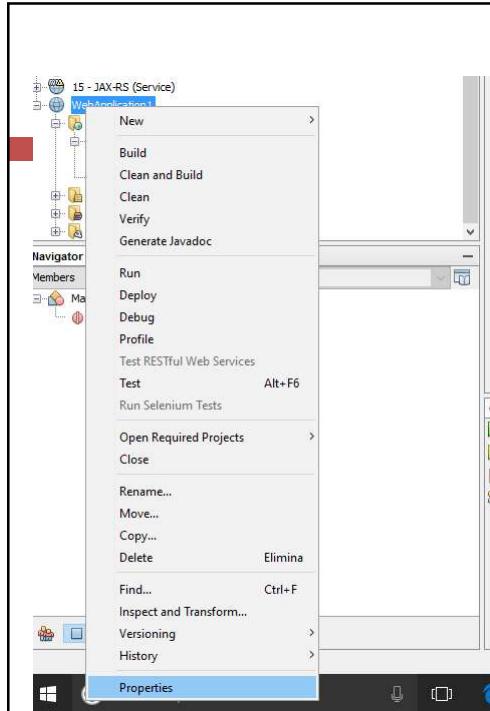
37

- Scrivere una servlet che risponde a
http://localhost:8080/WebApplication_Lab1/NewServlet
- Che riceve, iniettata, una istanza di un BookService
- ... dal quale stampa (nella risposta HTML) il libro
- Partiamo dall'includere l'esempio del libro...
 - basta fare drag and drop
 - ...dal progetto del libro del package nel sorgente del nuovo progetto
- Risolvendo alcuni problemi: serve una libreria da includere per risolvere alcune dipendenze (right-click su progetto)
- Ovviamente è da cancellare la classe Main

37

Aggiunta Libreria

- Properties
- Right-click su WebApplication_Lab1

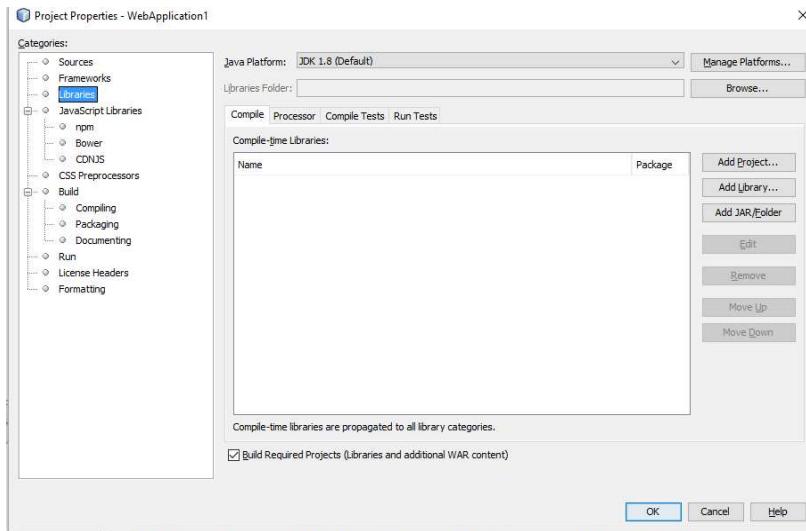


38

19

Aggiunta Libreria

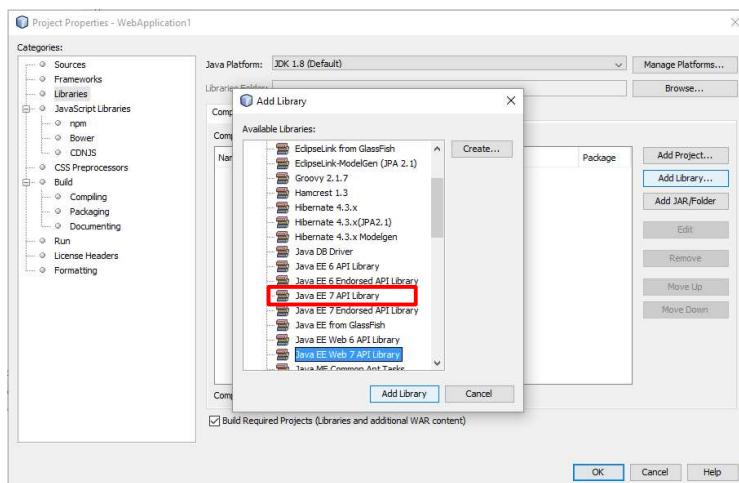
44



39

Aggiunta Libreria

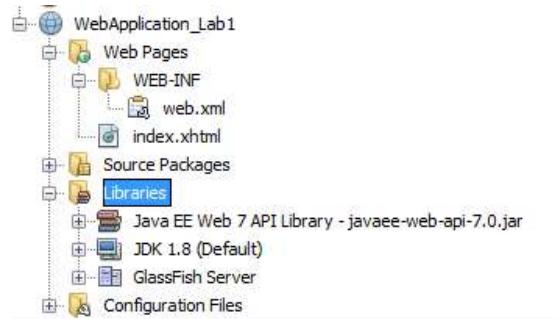
45



40

Aggiunta Libreria

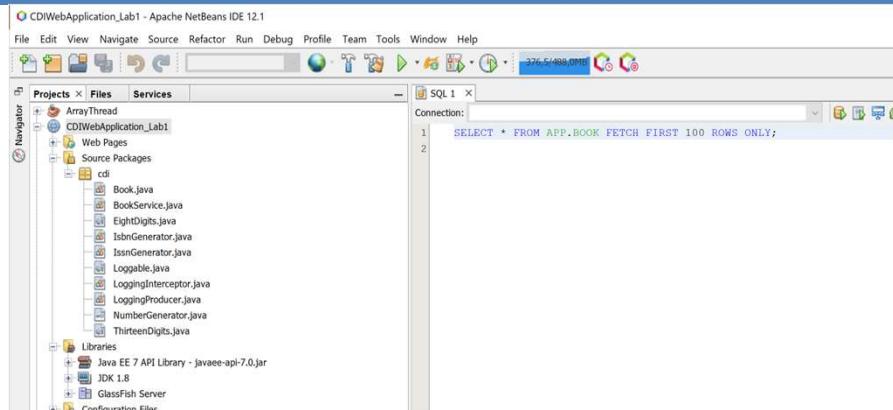
46



41

Il progetto

47



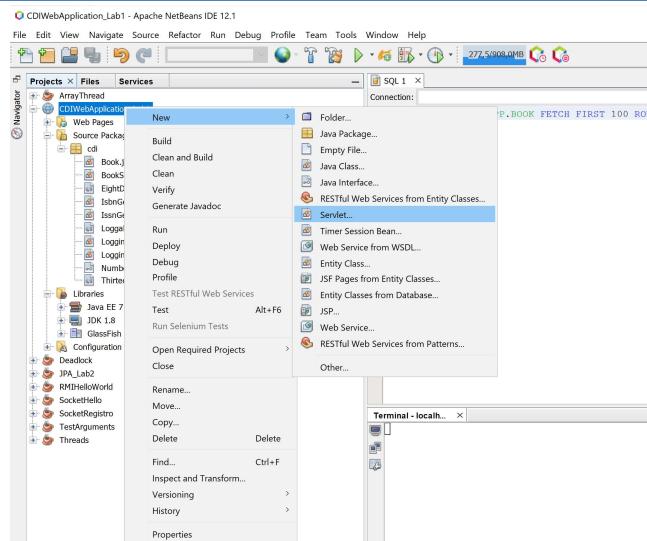
Scriviamo tutte le classi

Troverete il progetto sulla piattaforma

42

Creazione della Servlet - 1

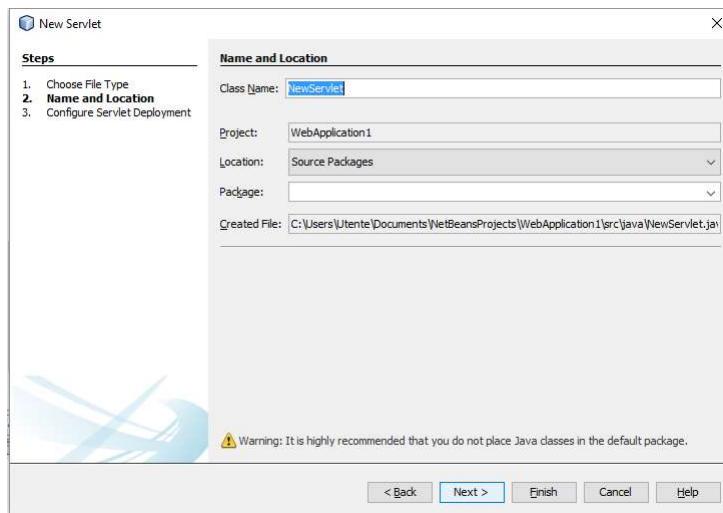
48



43

Creazione della Servlet - 2

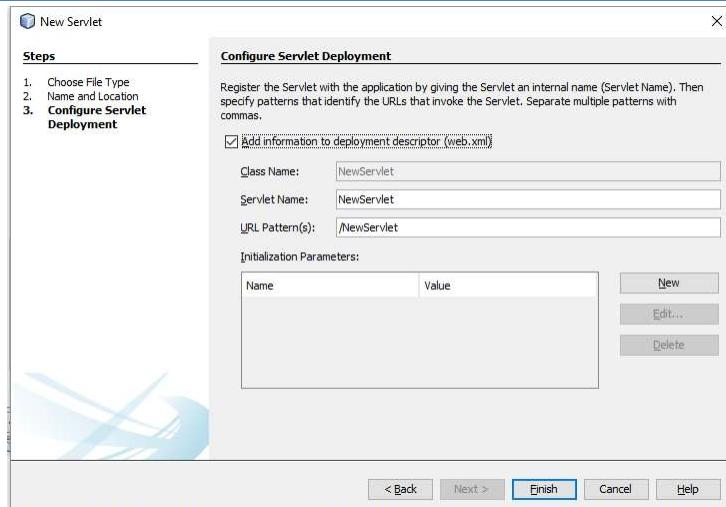
49



44

Creazione della Servlet - 3

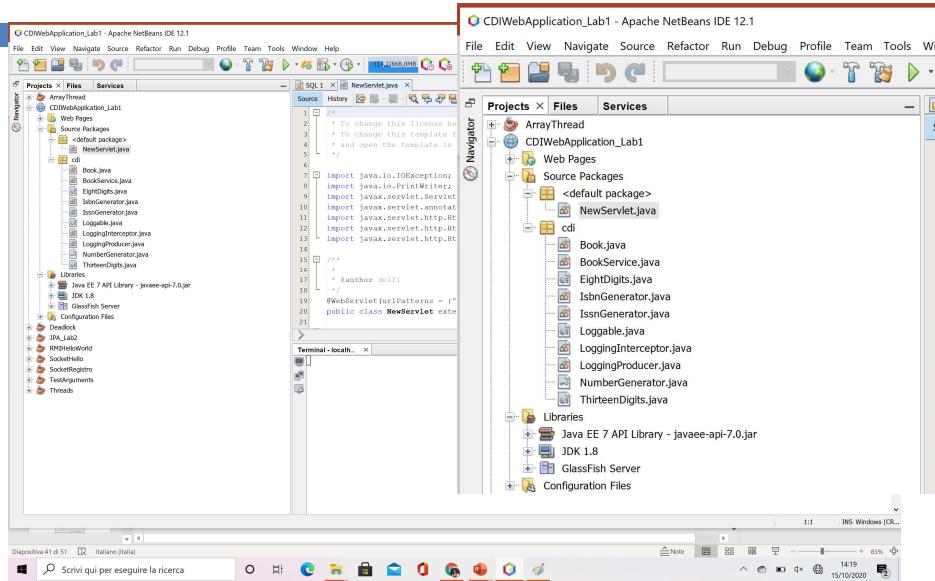
50



45

Creazione della Servlet - 4

51

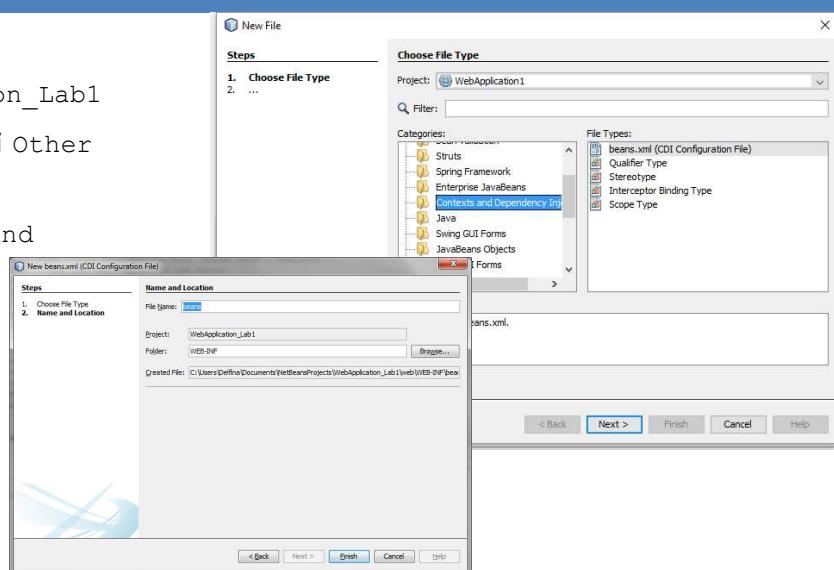


46

File beans.xml

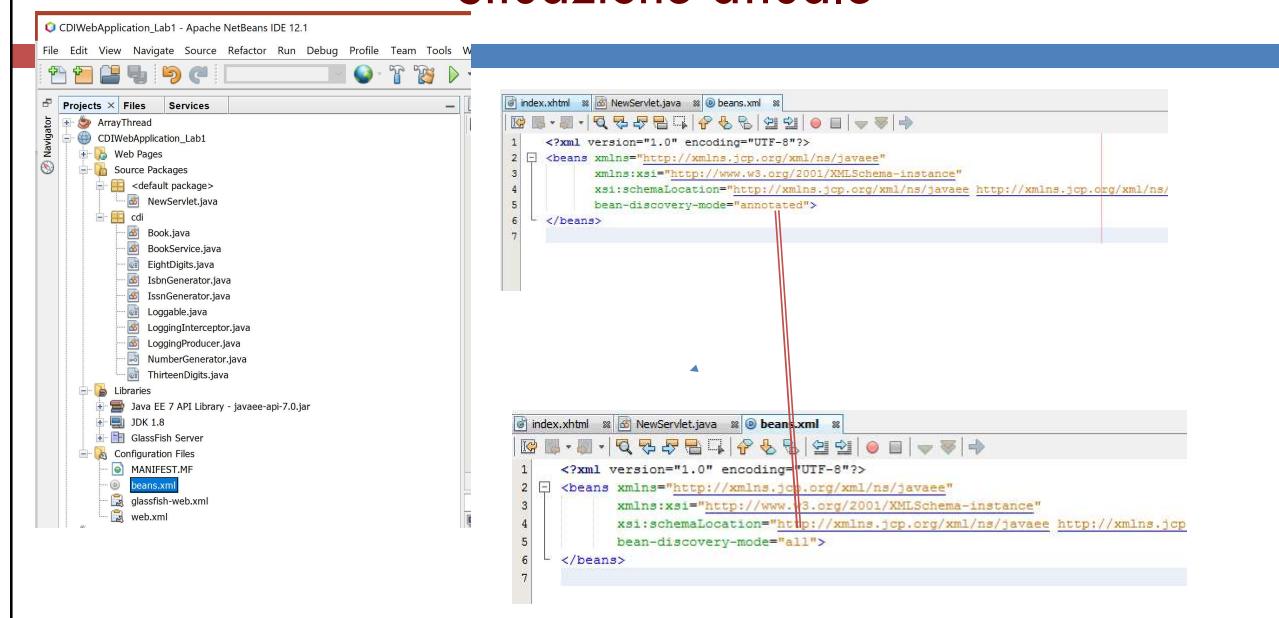
52

- Right-click sul Progetto WebApplication_Lab1
- Selezionare New e quindi Other
- Dalla lista selezionare Contexts and Dependency Injection



47

Situazione attuale



48

Come è fatta la Servlet - 1

```

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class NewServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
                                   HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try(PrintWriter out = response.getWriter()) {
            /* TODO output your page here. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>ServletNewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>ServletNewServletat" +
                       request.getContextPath() +
                       "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}

```

- › Import
- › Estende classe astratta che fornisce metodi già implementati
- › Metodo interno, unico per gestire sia GET che POST
- › Parametro risposta
- › Stream bufferizzato di output
- › Si scrive il file HTML ...
- › ... fino alla fine

49

Come è fatta la Servlet - 2

```

55
//...
@Override
protected void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

@Override
protected void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
}

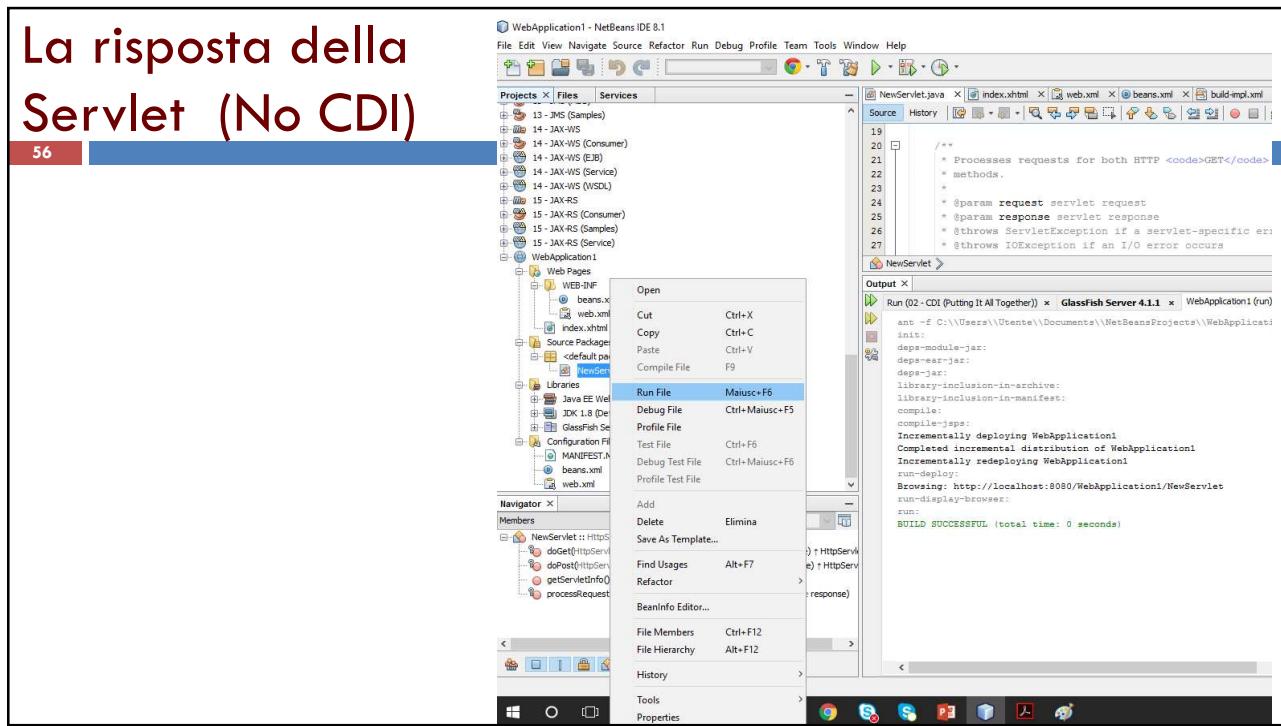
```

- › Metodo per trattare GET
- › ... con i parametri
- › ... passati a metodo interno unico
- › Stesso per POST

50

La risposta della Servlet (No CDI)

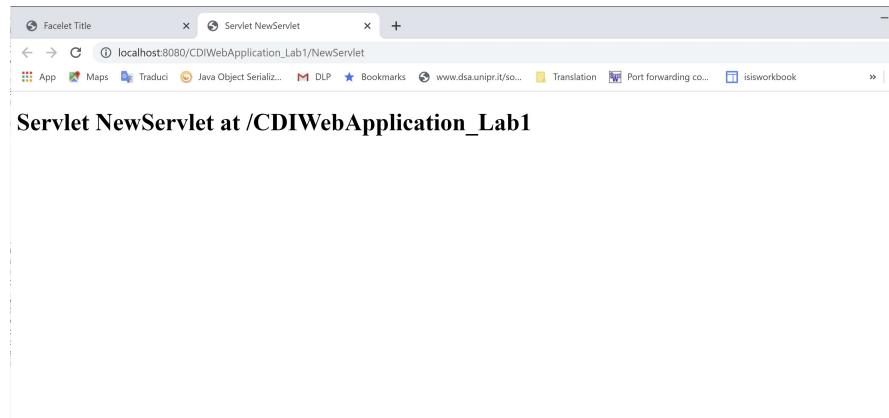
56



51

La risposta della Servlet (No CDI)

57



52

Mettiamo insieme il tutto!

58

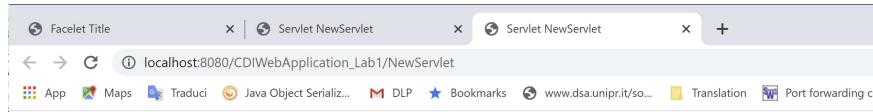
```
//...altriimport
import org.agoncal.book.javaee7.chapter02.*;
public class NewServlet extends HttpServlet {
    @Inject
    BookService b;
    protected void processRequest(HttpServletRequest request,
                                   HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try(PrintWriter out = response.getWriter()) {
            /* TODO output your page here. */
            out.println("<!DOCTYPE html>");
            out.println("<html>"); out.println("<head>");
            out.println("<title>ServletNewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>ServletNewServletat"+request.getContextPath()+"</h1>");
            Book book = b.createBook("Cambiare l'acqua ai fiori",
12.5f,"Romanzo");
            out.println("<h3>Libro creato:"+book+"</h3>");
            out.println("</body>");
            out.println("</html>");
        }
    }
//...
```

- › Import classi esempio
- › Servlet
- › Risorsa iniettata dal container
- › Inizio output
- › Creiamo un libro dal servizio offerto dal container
- › Lo stampiamo a video
- › Fine dell'output

53

Il risultato

59



54

Organizzazione della lezione

55

- Setup dell'ambiente di sviluppo
 - Netbean 8, GlassFish v. 4.1.1
- Come caricheremo i progetti
- Primo esempio (CDI)
 - Application server standalone
- Secondo esempio (Computazione lato Server HTTP)
- Conclusioni