JAVA – J2EE Programmer

Architetture distribuite

Applicazioni distribuite

- Un'applicazione costituita da diversi componenti, che viene eseguita in ambienti separati, normalmente su diversi calcolatori connessi in rete.
- Lo scambio messaggi è il paradigma di comunicazione più semplice per lo sviluppo di applicazioni distribuite
  - Ogni componente dell’applicazione possiede uno o più indirizzi
  - Un componente A comunica con un componente B spedendo un messaggio ad uno degli indirizzi associati a B
- In base ai livelli di elaborazione coinvolti, si distinguono in :
  - two-tier
  - three-tier
  - multi-tier

Applicazioni distribuite

- Un‘ **applicazione two-tier o client-server** è un tipo di applicazione di rete nel quale un client istanzia l'interfaccia di un'applicazione connettendosi ad una server application o ad un sistema di database
- Componenti distinti in due tipi: client e server
  - I server erogano un servizio
  - I client sfruttano tale servizio
- Esempio: applicazione web
  - Client: il browser
  - Server: il demone http che fornisce i documenti ai client

Applicazioni distribuite

- Un‘ **applicazione three-tiers** è un tipo di applicazione di rete in cui viene effettuare una divisione in 3 livelli: presentazione, logica e dati
- Componenti distinti in tre tipi: presentation, logic e data tier
  - Presentation tier: si occupa di visualizzare il risultato all’utente
  - Logic tier: stabilisce ed esegue la logica di elaborazione
  - Data tier: Il database detiene i dati
- Esempio: applicazione web
  - Web Server: si occupa di servire le pagine statiche
  - Application Server: si occupa di elaborare le pagine dinamiche
  - DataBase: gestisce e consente l’accesso ai dati

Applicazioni distribuite

- Un‘ **applicazione n-tiers** è un tipo di applicazione di rete in cui viene effettuare una divisione in più livelli
- Componenti: presentation, process managements, middleware, logic e data tier
  - Presentation tier: si occupa di visualizzare il risultato all’utente
  - Middleware tier: si occupa di intermediare la comunicazione
  - Process Management: stabilisce la logica del processo
  - Logic tier: esegue la logica di elaborazione
  - Data tier: Il database detiene i dati
- Esempio: applicazione web
  - Web Server: si occupa di servire i le pagine statiche
  - Application Server: si occupa di elaborare le pagine dinamiche
  - DataBase: gestisce e consente l’accesso ai dati

Applicazioni distribuite

- **Word Wide Web (WWW)** è stato proposto nel 1989 da Tim Berners-Lee. ! L'idea alla base del progetto era quella di fornire strumenti adatti alla condivisione di documenti statici in forma ipertestuale disponibili su Internet. Il Web segue un **modello Client/Server**
- I Client
  - utilizzano il protocollo http per connettersi ai server
  - Richiedono pagine web ai server e nel visualizzano il contenuto
  - I client sono tipicamente web browser, es IE, Mozilla., etc.
- I Server
  - Utilizzano il protocollo http per interagire con i client
  - Rimangono in ascolto di eventuali connessioni di nuovi client
  - Forniscono ai client le pagine web che questi richiedono

Applicazioni distribuite

HTTP: Hyper Text Transfer Protocol

- HTTP è un protocollo applicativo basato sul protocollo TCP
- Sia richieste al server, sia le risposte ai client sono trasmesse usando stream TCP

Applicazioni distribuite

Uniform Resource Identifier

- Forniscono un meccanismo semplice ed estensibile per identificare una risorsa
- Per **risorsa** intendiamo qualunque cosa che abbia una identità.
- **Esempi** sono un documento, una immagine, un servizio, una collezione di risorse.

Applicazioni distribuite

Le URI possono essere classificate in:

- **Uniform Resource Locator (URL)**
  - Il termine URL riferisce il sottoinsieme delle URI che identificano le risorse per mezzo del loro meccanismo di accesso primario (es. la loro locazione nella rete)
- **Uniform Resource Name (URN)**
  - Il termine URN riferisce il sottoinsieme delle URI che devono rimanere globalmente uniche e persistenti anche qualora la risorsa cessi di esistere o diventi non disponibile.
  - Esempio urn:isbn:0-395-36341-1 stabilisce il sistema di identificazione International Standard Book Number e l'idenficatore del libro ma non dice come ottenerne una copia.

Applicazioni distribuite

HTTP: Hyper Text Transfer Protocol

- **Prevede la presenza di un determinato scenario**
  - **Client**: Programma applicativo che stabilisce una connessione al fine di inviare delle Request
  - **Server**: Programma applicativo che accetta connessioni al fine di ricevere Request ed inviare specifiche Response con le risorse richieste.
  - **Connessione**: circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione
  - **Messaggio**: è l'unità base di comunicazione HTTP

Applicazioni distribuite

Un messaggio HTTP è definito da due strutture:

- **Message Header**: Contiene tutte le informazioni necessarie per la identificazione del messaggio
- **Message Body**: Contiene i dati trasportati dal messaggio.

Applicazioni distribuite

Metodi HTTP utilizzabili:

- **GET**: richiedo una specifica risorsa attraverso un singolo URL. Posso passare diversi parametri, la lunghezza massima di un URL è limitata
- **POST**: richiedo una specifica risorsa evidenziando che il body del messaggio contiene i dettagli per la identificazione e la elaborazione della risorsa stessa: non ci sono limiti di lunghezza nei parametri di una richiesta
- **DELETE**: richiedo la cancellazione della risorsa riferita dall'URL specificato
- **PUT**: richiedo che il documento allegato sia memorizzato all'URL specificato
- **OPTIONS**: rappresenta la richiesta di informazioni sulle opzioni disponibili per la comunicazione.

Applicazioni distribuite

Metodi HTTP utilizzabili:

- **TRACE**: è usato per invocare il loop-back remoto a livello applicativo del messaggio di richiesta. Consente al client di vedere cosa è stato ricevuto dal server ed ha applicazione nella diagnostica e nel testing dei servizi web.
- **HEAD**: è simile al metodo GET. A seguito di una HEAD il server restituisce solo lo header del messaggio di risposta.

Applicazioni distribuite

Esempio di Request HTTP

GET /search?q=Web+Technologies HTTP/1.1

Host: www.google.com

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.2) Gecko/20040803

Accept: text/xml,application/xml,application/xhtml+xml,

text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

Accept-Language: da,en-us;q=0.8,en;q=0.5,sw;q=0.3

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Referer: http://www.google.com/

Applicazioni distribuite

Esempio di Response HTTP

HTTP/1.1 200 OK

Date: Fri, 17 Sep 2009 07:59:01 GMT

Server: Apache/2.0.50 (Unix) mod_perl/1.99_10 Perl/v5.8.4 mod_ssl/2.0.50 OpenSSL/0.9.7d DAV/2 PHP/4.3.8 mod_bigwig/2.1-3

Last-Modified: Tue, 24 Feb 2009 08:32:26 GMT

ETag: "ec002-afa-fd67ba80"

Accept-Ranges: bytes

Content-Length: 2810

Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>...</html>

Applicazioni distribuite

Status Code

- 1xx Informational : Da http 1.0 in poi non devono essere usati se non in condizioni di test.
- 2xx Success: la richiesta del client è stata ricevuta, e processata con successo
- 3xx Redirection: il client deve intraprendere ulteriori azioni per completare la request.
- 4xx Client Error: errore nella request da parte del client
- 5xx Server Error: il server ha subito un errore nel processare una request apparentemente valida

Applicazioni distribuite

Status Code

- 1xx Informational : Da http 1.0 in poi non devono essere usati se non in condizioni di test.
- 2xx Success: la richiesta del client è stata ricevuta, e processata con successo
- 3xx Redirection: il client deve intraprendere ulteriori azioni per completare la request.
- 4xx Client Error: errore nella request da parte del client
- 5xx Server Error: il server ha subito un errore nel processare una request apparentemente valida

Piattaforma J2EE

Piattaforma J2EE

J2EE Java 2 Enterprise Edition

- La tecnologia J2EE realizzata su solidissime basi, rivoluzionò le tecnologie di sviluppo del software svariati anni or sono.
- La tecnologia J2EE viene utilizzata per lo sviluppo di applicazioni aziendali e sistemi di elaborazione in generale.
- Si chiamano Applicazioni “Enterprise” in quanto:
  - supportano i processi di business
  - distribuite in rete (solaris, linux, windows)
  - mettono in relazione dipartimenti e unità operative
  - multilivello (n-tier)
  - operano in ambienti eterogenei (pc, mainframe, server)
  - comunicano con protocolli diversi (HTTP, JMS)
  - Supporto agli standard (HTTP, XML HTML, SOAP)

Piattaforma J2EE

Piattaforma J2EE

Piattaforma J2EE

Le 4 “S”

- Una applicazione efficace devono supportare le 4 S:
  - Stabilità
  - Scalabilità
  - Sicurezza
  - Semplicità

Piattaforma J2EE

Stabilità

- La stabilità è la proprietà di un’applicazione quando si comporta come previsto senza bloccarsi e senza subire crash
- La stabilità è particolarmente importante in applicazioni mission-critical
- Le applicazioni instabili possono influenzare negativamente un’organizzazione
  - utilizzando risorse eccessive
  - sprecando denaro per l’addizionale assistenza tecnica richiesta

Piattaforma J2EE

Scalabilità

- La scalabilità è l’attitudine di un’applicazione a poter essere facilmente ampliata, soddisfacendo eventuali incrementi di carico nelle richieste.
- Scalabilità limitata indica che una applicazione non risulta disponibile in condizioni di carico particolarmente gravose
- Scalabilità elevata indica che una applicazione è in grado di adattarsi dinamicamente all’incremento del carico

Piattaforma J2EE

Sicurezza

- La sicurezza è il grado di protezione o vulnerabilità di un’applicazione da utilizzazioni non autorizzate
- Oggi è una questione di estrema importanza per molte organizzazioni e individui
- Certamente aumentano i costi di sviluppo e i tempi di consegna, ma nel lungo termine, questi investimenti iniziali generano ritorni considerevoli, aumentando la soddisfazione e la confidenza dei clienti

Piattaforma J2EE

Semplicità

- La semplicità di un’applicazione dipende dal grado di semplicità della stessa
  - dalla prospettiva degli utenti finali
  - dalla prospettiva dei programmatori
- Per gli utenti finali dipende fondamentalmente dall’interfaccia utente con la quale interagiscono
- Per lo sviluppatore software dipende dalla possibilità di sviluppare o migliorare facilmente ed efficientemente l’applicazione, mantenendone la stabilità, la scalabilità e la sicurezza globali

Piattaforma J2EE

Componenti

- L’architettura di J2EE è basata sulle componenti
- Pertanto le applicazioni J2EE consistono quindi di componenti software auto-contenute che vengono assemblate per ottenere un’applicazione enterprise distribuita
- Ognuna di esse risiede nel particolare livello dell’ambiente di elaborazione. J2EE supporta esplicitamente quattro livelli:
    - funzionalità client
    - supporto Web
    - logica di business
    - integrazione con database.

Piattaforma J2EE

Modello classico di applicazione client-server a due livelli

L'architettura multilivello di J2EE è molto flessibile nel partizionamento logico e fisico dei livelli.

Piattaforma J2EE

Componenti

- L’architettura di J2EE è basata sulle componenti
- Pertanto le applicazioni J2EE consistono quindi di componenti software auto-contenute che vengono assemblate per ottenere un’applicazione enterprise distribuita
- Ognuna di esse risiede nel particolare livello dell’ambiente di elaborazione. J2EE supporta esplicitamente quattro livelli:
    - funzionalità client
    - supporto Web
    - logica di business
    - integrazione con database.

Piattaforma J2EE

Componenti

- Codice riutilizzabile :
  - le componenti J2EE vengono assemblate fra di loro per formare delle applicazioni complete, consentendo così agli sviluppatori di comporre queste unità di funzionalità riutilizzabili in modo modulare a seconda delle esigenze tipo “LEGO”
- Facilita la manutenzione e l’estensione
  - Le architetture basate su componenti generano applicazioni modulari che sono relativamente semplici da mantenere, aggiornare ed estendere
- Supporta il packaging e il deployment delle componenti
  - Packaging e il deployment delle componenti può essere effettuato singolarmente, collettivamente con librerie di componenti o globalmente nella forma di applicazioni

Piattaforma J2EE

Componenti

- Supporto built-in per la scalabilità e la sicurezza
  - Le componenti J2EE sono sempre eseguite nel contesto di un container
  - E’ previsto la gestione dei privilegi di accesso su base individuale e di gruppo
- Sviluppato su basi stabili e affidabili
  - solide basi offerte dalla piattaforma J2SE sottostante e dal linguaggio di programmazione Java con il quale è realizzata

Piattaforma J2EE

Componenti

- Bytecode platform-independent
  - i programmi Java consistono in codice “bytecode” indipendente dalla specifica architettura hardware
- Integrazione con sistemi informativi aziendali e sistemi legacy
  - intenzionalmente progettata per poter integrare sistemi EIS e legacy preesistenti grazie alla varietà di API supportate e dallo specifico orientamento della piattaforma all’interoperabilità.

Piattaforma J2EE

API

- L’API J2EE Connector Architecture:
  - fornisce un framework che consente l’integrazione e il packaging di resource adapter (“adattatori di risorsa”) con qualsiasi applicazione J2EE.
  - I resource adapter sono essenzialmente driver software che supportano la connettività con uno specifico tipo di sistema informativo aziendale.
- L’API JDBC:
  - fornisce alle applicazioni Java l’accesso a sorgenti di dati tabellari quali
    - database SQL (Structured Query Language)
    - fogli di calcolo
    - file che contengono tabelle di dati

Piattaforma J2EE

API

- L’API JNDI (Java Naming and Directory Interface):
  - fornisce funzionalità di naming (“nominazione”) e di directory ai programmi Java.
- L’API JMS (Java Message Service):
  - fornisce alle applicazioni J2EE l’accesso a sistemi di messaging aziendaleche consentono alle applicazioni di comunicare con scambio asincrono di messaggi
- L’API JavaMail:
  - consente di inviare e ricevere messaggi e-mail

Piattaforma J2EE

API

- L'API JTA (Java Transaction API):
  - fornisce funzionalità transazionali alle applicazioni Java.
  - Una transazione è un'unità di lavoro atomica composta da più compiti che
    - o sono tutti completati
    - o non ne viene concluso alcuno e deve essere ripristinata la situazione antecedente al tentativo di esecuzione
- Le API Java IDL e RMI-IIOP:
  - consentono alle applicazioni J2EE di accedere a sistemi basati su CORBA (Common Object Request Broker Architecture).
  - CORBA definisce uno standard per la comunicazione fra applicazioni indipendentemente da dove esse risiedano o dal linguaggio in cui siano state scritte.

Piattaforma J2EE

API

- JAXP (Java API for XML Processing):
  - fornisce le funzionalità fondamentali di elaborazione dei documenti XML, inclusa la possibilità di effettuare il parsing di documenti XML utilizzando il parser built-in
    - DOM (Document Object Model)
    - il SAX (Simple API for XML Parsing)
    - un altro parser da fornire
- JAX-RPC (Java API for XML-based RPC):
  - fornisce il supporto per le chiamate di procedura remota (RPC) dei servizi Web attraverso il protocollo SOAP/HTTP

Piattaforma J2EE

API

- SAAJ (SOAP with Attachments API for Java):
  - fornisce alle applicazioni la possibilità di creare, inviare, accedere e manipolare messaggi SOAP
- JAXR (Java API for XML Registries):
  - fornisce ai client J2EE l’accesso a server di registro basati su XML, come:
    - UDDI (Universal Description, Discovery and Integration)
    - ebXML.

Piattaforma J2EE

Design pattern adottatti in J2EE

- Il programma Enterprise BluePrints ha documentato i seguenti 13 pattern nel J2EE :
  - **Business Delegate**: riduce l'accoppiamento fra i livelli Web e EJB.
  - **Composite Entity**: modella reti di entità business relazionate.
  - **Composite View**: consente di creare delle view ("viste") dell'applicazione da altre view riutilizzabili.
  - **Data Access Object (DAO)**: permette di modificare i meccanismi di accesso ai dati indipendentemente dal codice che utilizza i dati.
  - **Fast Lane Reader**: fornisce un meccanismo efficiente per accedere a dati tabulari di sola lettura.

Piattaforma J2EE

Design pattern adottatti in J2EE

- **Front Controller**: centralizza le richieste di elaborazione delle applicazioni attraverso una singola componente controller.
- **Intercepting Filter**: consente la pre-elaborazione e la post-elaborazione, ovvero il filtering, delle richieste delle applicazioni.
- **Model-View-Controler (MVC)**: disaccoppia il comportamento, la rappresentazione e la presentazione dei dati delle applicazioni.
- **Service Locator**: semplifica l'accesso ai servizi business centralizzando la ricerca degli oggetti dei servizi distribuiti.
- **Session Facade**: centralizza e coordina le operazioni fra gli oggetti business coinvolti in un workflow.

Piattaforma J2EE

Design pattern adottatti in J2EE

- **Transfer Object**: consente il trasferimento dei dati business fra i vari livelli.
- **Value List Handler**: fornisce un meccanismo efficiente per iterare su liste voluminose di sola lettura distribuite tra più livelli.
- **View Helper**: disaccoppia le classi business e applicazione per semplificare l’accesso allo stato del modello e facilitare l’accesso alla logica dei dati per la view del-l’applicazione

Piattaforma J2EE

Vendor J2EE

- Numerosi vendor di prodotti supportino la piattaforma J2EE, per la quale nel corso degli anni si è potuto sviluppare un fiorente mercato competitivo.
  - Jboss di Jboss Group
  - Tomcat di Apache
  - WebSphere Technology
  - WebLogic Server di BEA
  - JRun di Macromedia Application
  - Sun ONE Studio
  - Oracle9iJDeveloper di Oracle
  - Jbuilder di Borland
  - JProbe Suite di Sitraka

Java Servlet

Diffusione di Internet

- Negli anni ‘90 la crescente diffusione di internet viene supportata da vari fattori:
  - Affermazione del linguaggio HTML per la realizzazione dei siti web in modalità statica
  - Diffusione estensioni dinamiche D-HTML
  - Linguaggi di scripting a supporto di pagine dinamiche
  - Sviluppo di nuove tecnologie web (Servlet, Applet, ASP)
- Tecnologie Web lato:
  - Client
  - Server
  - Miste

Tipologie di tecnologie

- Tecnologie lato client:
  - Nelle tecnologie lato client la dinamicità e l’interazione con l’utente sono gestite direttamente da codice eseguito dal client o da un suo plugin.

  Attraverso:
  - Linguaggi di scripting: Javascript , VBscript, Jscript
  - Plugins dei Browser: JVM per esecuzione di Applet
- Tecnologie lato server:
  - Nelle tecnologie lato server, il client ha un ruolo essenzialmente passivo, ed è il server a gestire la parte dinamica.

  Attraverso
  - Common Gateway Interface, le Active Server Pages (ASP) di Microsoft e, le Servlet e le Java Server Pages. Client

Tipologie di tecnologie

- Tecnologie miste:
  - uso di plugin sia sul lato client che lato server che interagiscono direttamente tra loro scavalcando il normale canale di comunicazione client–server basato su HTTP

  Attraverso:
  - L’uso congiunto di Applet e Servlet Java.
  - Real Player, Macromedia Flash, ecc., i quali consentono al server di inviare e al client di ricevere, suoni e animazioni

Tecnologie lato Server

- Assume particolare rilevanza nell’ambito delle applicazioni business, quello delle tecnologie “server side", basate sul protocollo HTTP per lo sviluppo di applicazioni web centrate sulla gestione di basi di dati.
- Tecnologie più diffuse:
  - CGI (Common Gateway Interface)
  - Servlet e JSP

CGI
Common Gateway Interface

- CGI (Common Gateway Interface)
  - È un’interfaccia per mezzo della quale il server web è in grado di lanciare delle applicazioni che risiedono sul server
  - Le applicazioni sono normali file eseguibili: sia un eseguibile binario che uno script interpretato direttamente dalla shell o da un altro interprete
  - Tutto ciò che l’applicazione scrive sullo standard output viene intercettato dal server e inviato al client.
- Svantaggio
  - consumo di risorse del sistema, dato che ogni esecuzione CGI dà origine a un nuovo processo

Java Servlet

Vantaggi delle Servlet rispetto alle CGI

- Maggiore efficienza
- Migliore interfacciamento
- Portabilità
- tutta la potenza e l’eleganza del linguaggio Java
- innumerevoli librerie standard ed estensioni standard.

Java Servlet

Maggiore efficienza

- Ogni volta che il client esegue una richiesta di un servizio basato su **CGI**, il web server deve mandare in esecuzione un processo dedicato per quel client. Se n client effettuano la stessa richiesta, allora **n processi devono essere prima istanziati ed eseguiti poi**. Questo comporta un notevole dispendio di tempo e di risorse di macchina.
- Un **Servlet** invece, una volta mandato in esecuzione può servire un numero n di richieste senza la necessità di ulteriori esecuzioni. Il server manda in esecuzione **una sola JVM**, la quale effettua un caricamento dinamico delle classi necessarie per eseguire i vari Servlet invocati.

Java Servlet

Maggiore efficienza

- Nel caso delle **Servlet** per ogni client che effettui una richiesta il server manderà in esecuzione un thread che eseguirà il metodo service (o, in maniera equivalente, il doGet() o doPost() a seconda della implemtenzione).
- Il tutto in modo **automatico e trasparente** agli occhi del programmatore che dovrà solo preoccuparsi di scrivere il codice relativo alle operazioni da effettuare in funzione della richiesta di un client.
- Eventualmente si tenga presente che è possibile realizzare Servlet **monothread**, in grado di servire un client per volta.

Java Servlet

Migliore interfacciamento

- La separazione fra i metodi init() e service() permette di:
  - ottimizzare le risorse
  - migliorare la modalità di interfacciamento con il client.
- Il metodo init() serve per inizializzare il Servlet, ed è il posto dove tipicamente si eseguono le operazioni computazionalmente costose da effettuare una volta per tutte (come la connessione a un DB).
- Il metodo service() invece è quello che viene mandato in esecuzione al momento di una chiamata POST o GET.

Java Servlet

Portabilità

- Poiché Java è un linguaggio interpretato e non compilato è facilmente eseguibile su piattaforme eterogenee senza la necessità di dovere essere nuovamente ricompilato .
- L’unica esigenza è la presenza della JVM e del container che gestisce il suo ciclo di vita.

Java Servlet

- Uno dei settori sul quale si focalizza **maggiormente l'attenzione** dello scenario della Information Technology è quello :
  - **della programmazione web oriented**, ovvero quella in cui la parte client è costituita da un semplice browser che interagisce con la parte server per mezzo del protocollo HTTP.
- **Sequenza** delle operazioni web:
  - Il client tramite browser effettua una chiamata al web server;
  - il web server provvede a eseguire l'applicazione;
  - l'applicazione dopo aver eseguito tutte le operazioni del caso, produce un output che viene passato al web server che lo invierà poi al client.
- Quando Sun introdusse le Servlet API, diventarono in poco tempo una delle più importanti di tutta la tecnologia Java.

Java Servlet

La Servlet API

- Un Servlet è “**componente lato server per l’estensione di web server Java enabled**”,
- Un Servlet è quindi un programma Java in esecuzione sul server ed in grado di colloquiare con il client per mezzo del protocollo HTTP.
- Tipicamente questo si traduce nella possibilità di generare dinamicamente contenuti web da visualizzare nella finestra del client/browser.
- I packages che contengono tutte le classi necessarie per la programmazione dei Servlet sono il
  - javax.servlet
  - javax.servlet.http
- Si tratta di Standard Extension API, ovvero non fanno parte del JDK core.

Java Servlet

Container

- Un container è un oggetto all’interno del quale il Servlet vive e nel quale trova un contesto di esecuzione personalizzato.
- Il concetto di container ha poi una visione più ampia dato che viene adottato anche nel caso di JSP ed EJB.

Java Servlet

Ciclo di vita della Servlet

- Un Servlet segue un suo ciclo di vita ben preciso, composto dalla
  - Inizializzazione
  - gestione delle invocazioni da parte dei client
  - disintallazione.
- Ognuna di queste fasi è particolarmente importante, dato che condiziona
  - la logica di funzionamento complessiva
  - le performance dell’applicazione nel suo complesso.

Java Servlet

Java Servlet

```
public class MiaServlet extends HttpServlet {

    public void init() {
        super.init();
    }
    public void service (HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html><head></head><body>Hello World!
    </body></html>");
    }
    public void destroy() {
        super.destroy();
    }
}
```

Java Servlet

Ciclo di vita della Servlet

- Per realizzare un Servlet HTTP è sufficiente
  - **estendere la classe HttpServlet**, appartenente al package javax.servlet.http
  - **ridefinire i metodi**:
    - **init**(): per personalizzare l'inizializzazione del Servlet al container. Per esempio aprire la connessione al DataBase
    - **service**(), **doPost**() e **doGet**() : per definire invece il comportamento del Servlet in funzione delle invocazioni del client.
    - **destroy**(): per personalizzare la cancellazione del Servlet nel container. Per esempio chiudere la connessione al DataBase

Java Servlet

Inizializzazione di un Servlet

- Il metodo **init**(), derivato dalla classe GenericServlet, ha lo scopo di
  - effettuare tutte quelle operazioni necessarie per l'inizializzazione del Servlet stesso
  - per il corretto funzionamento successivo.
- La firma del metodo è la seguente:

  public void **init** (ServletConfig config) throws ServletException

Java Servlet

Errore in fase di inizializzazione gestite dal container:

- **ServletException** : se durante il processo di inizializzazione del Servlet si verifica un errore allora il Servlet potrà segnalare tale evento generando una eccezione di tipo ServletException.
- In questo caso il Servlet non verrà reso disponibile per l'invocazione, e **l'istanza** appena creata verrà immediatamente **rilasciata**.
- Il metodo **destroy**() in questo caso **non verrà invocato dal server**, considerando il Servlet non correttamente inizializzato.
- Dopo il rilascio dell'istanza fallita, il server procederà immediatamente, alla **istanziazione ed inizializzazione di un nuovo Servlet;**
- La **UnavailableException** permette di specificare il **tempo minimo** necessario da attendere prima di intraprendere nuovamente il processo di inizializzazione.

Java Servlet

Il resto della vita: i metodi doGet(), doPost() e service() :

- Dopo l’inzializzazione, un Servlet si mette in attesa di una eventuale chiamata da parte del client, che potrà essere
  - una GET HTTP
  - una POST HTTP.
- L’interfaccia Servlet mette a disposizione a questo scopo i metodi:
  - public void **service** (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
  - protected void **doGet** (HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
  - protected void **doPost** (HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

Java Servlet

Il resto della vita: i metodi doGet(), doPost() e service() :

- Il server per ogni invocazione da parte del client manda in esecuzione un thread separato.

- Il metodo **service()**
  - viene invocato indistintamente sia nel caso di una invocazione tipo GET che POST. La ridefinizione del metodo service permette di definire il comportamento del Servlet stesso.
- I metodo **doGet**() e **doPost**()
  - In caso contrario viene eseguito il metodo doGet() o doPost() a seconda che sia stata fatta una chiamata HTTP di tipo Get o Post

Java Servlet

I parametri in ingresso:

- I due parametri passati sono
  - **HttpServletRequest**
  - **HttpServletResponse**
- che permettono di interagire con la richiesta effettuata dal client e di inviare risposta tramite pacchetti HTTP.

Java Servlet

Come interagire con i Servlet: richieste e risposte :

- Un Servlet può comunicare con il client in maniera bidirezionale per mezzo delle due interfacce
  - **HttpServletRequest** : rappresenta la richiesta e contiene i dati provenienti dal client
  - **HttpServletResponse**: rappresenta la risposta e permette di incapsulare tutto ciò che deve essere inviato indietro al client stesso.
- La **HttpServletRequest** permette di ricavare i parametri passati insieme all'invocazione del client tramite il metodo **getParameter()** è possibile ottenere il valore di un parametro dato il nome :

```
public String getParameter(String name)
```

Java Servlet

Java Servlet

```
<html>
 <head><title>MiaForm</title></head>
 <body>
    <form method="get" action="/MiaServlet">
          <input type="text" name="nome“ >
          <input type="submit“ value=“Invia”
    </form>
</body>
</html>
public class MiaServlet extends HttpServlet {
   public void service(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {
      PrintWriter out = res.getWriter();
       String nome = req.getParameter("nome");
       out.println("<html><head></head><body>Ciao " + nome + "</body></html>");
   }
}
```

Java Servlet

Come interagire con i Servlet: richieste e risposte :

- Per ottenere invece tutti i nomi o tutti i valori dei parametri si possono utilizzare i metodi:

  public String[] **getParameterValues**(String name)

  public Enumeration **getParameterNames**()

- Nel caso di una GET HTTP il metodo **getQueryString**()restituisce una String con tutti i parametri concatenati.
- Nel caso in cui si attendano dati non strutturati e in forma testuale, e l'invocazione sia una POST si può utilizzare il metodo **getReader**(), che restituisce un BufferedReader.
- Se invece i dati inviati sono in formato binario, allora è indicato utilizzare il metodo **getInputStream()**, il quale a sua volta restituisce un oggetto di tipo ServletInputStream.

Java Servlet

Come interagire con i Servlet: richieste e risposte :

- Per conoscere la collocazione sul file system di un dato URLsi usa il metodo **getRealPath()**

  ServletContext.getRealPath(String path)

Java Servlet

Gli **headers** associati alla chiamata

- La codifica dei **pacchetti HTTP** prevede una parte di intestazione detta header dove si possono codificare le informazioni legate alla tipologia di trasmissione in atto.
- Un Servlet può accedere all'header della **request HTTP** per mezzo dei seguenti metodi della interfaccia HttpServletRequest:
  - public String **getHeader**(String name): permette di accedere all'header dato il nome.
  - public Enumeration **getHeaderNames**() con la versione 2.2 della API è stato aggiunto anche il metodo che permette di ricavare headers multipli i quali possono essere presenti come nel caso nel **cache control**.

Java Servlet

Gli headers associati alla chiamata

- In alcuni casi gli header possono contenere informazioni di tipo
  - Stringa
  - Numerico
  - Data
- in questo caso può essere comodo utilizzare direttamente i metodi che restituiscono direttamente il tipo più opportuno
  - public int **getIntHeader**(String name)
  - public long **getDateHeader**(String name)
- Nel caso non sia possibile la conversione a:
  - Tipo numerico, verrà generata una **NumberFormatException**
  - Tipo data: allora l'eccezione generata sarà una **IllegalArgumentException**.

Java Servlet

Servlet e l’internazionalizzazione

- La possibilità di rendere una applicazione sensibile alla localizzazione geografica in cui viene eseguita è sicuramente una delle caratteristiche più interessanti che Java mette a disposizione grazie alla internazionalizzazione.
- Nelle Servlet 2.2 sono stati aggiunti alcuni metodi alla interfaccia HttpServletRequest con lo scopo di determinare la localizzazione del client e quindi agire di conseguenza.
  - **getLocale**() restituisce un oggetto di tipo java.util.Locale in base alle informazioni memorizzate nell header Accept-Language della richiesta.
  - **getLocales**() restituisce la lista di tutti i Locale accettati dal client con il preferito in testa alla lista.
  - **setLocale**() della HttpServletResponse permette di impostare il Locale opportuno per inviare la risposta nel modo migliore verso il client.

Java Servlet

Servlet e l’internazionalizzazione

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException,IOException {
 // scrive un output in base alla lingua  ricavata tramite locale.getLanguage()
 // che ritorna una stringa in base ad ISO 639
   res.setContentType("text/html");
   Locale locale = req.getLocale();
   String language = locale.getLanguage();
   String messagge = null;
   if (language.equals("it"))
      messagge = "Ciao";
   else if (language.equals("en"))
      messagge = "Hello";
   res.setLocale(locale);
   PrintWriter out = res.getWriter();
   out.print( message );
}
```

Java Servlet

Rispondere è cortesia: **HttpServletResponse**

- La risposta del Servlet può essere inviata al client per mezzo di un oggetto di tipo **HttpServletResponse** che offre due metodi per inviare dati al client:
  - **getWriter()** che restituisce un oggetto di tipo **java.io.Writer**, per inviare dati di tipo testuale
  - **getOutputStream()** che restituisce un oggetto di tipo **ServletOutputStream**, per inviare anche dati in forma binaria.
- Con il metodo **setContentType()** si può stabilire il tipo MIME della pagina di risposta.
- Con il metodo **setHeader()** si possono definire gliHEADER contenuti nella risposta.

Java Servlet

Caching della risposta:

- Prima del rilascio della API 2.2, molti server implementavano tecniche di buffering per migliorare le prestazioni (utilizzando memorie tampone tipicamente di circa 8 kilobyte)
- Adesso il cosiddetto response buffering è parte integrante della specifica ufficiale.
- Sono stati aggiunti alla interfaccia ServletResponse **cinque nuovi metodi** per la gestione diretta del buffer
  - sia che si utilizzi una comunicazione a mezzo di un **ServletOutputStream**
  - sia di un **Writer**.
- **getBufferSize() setBufferSize() isCommitted() flushBuffer() reset()**

Java Servlet

Caching della risposta:

- **getBufferSize()**
  - restituisce la dimensione del buffer attualmente utilizzata: nel caso in cui tale risultato sia un intero pari a zero, allora questo significa la mancanza di buffering.
- **setBufferSize()**
  - impostare la dimensione del buffer associata ad una risposta: E' sempre bene specificare una dimensione maggiore di quella utilizzata normalmente dal Servlet, al fine di
    - riutilizzare la memoria già allocata
    - velocizzare ulteriormente la modifica.
  - Deve essere invocato prima di ogni operazione di scrittura per mezzo di un **ServletOutputStream** o di un **Writer**.

Java Servlet

Caching della risposta:

- **isCommitted()**
  - restituisce un valore booleano ad indicare se siano stati inviati o meno dei dati verso il cliente.
- **flushBuffer()**
  - forza l’invio dei dati presenti nel buffer.
  - Una volta che il buffer viene riempito il server dovrebbe immediatamente effettuare una flush del contenuto verso il client
- **reset()**
  - provoca la cancellazione di tutti i dati presenti nel buffer.
  - Se il buffer è stato appena vuotato a causa dell invio della risposta verso il client, allora l invocazione del metodo reset() provocherà la generazione di una eccezione di tipo **IllegalStateException**.

Java Servlet

Esempio di caching

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException {
        // si imposta un buffer di 8 kb
        res.setBufferSize(8 * 1024);
        // restituisce la dimensione del buffer in questo caso 8192
        int size = res.getBufferSize();
        // si invia un messaggio nel buffer
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Primo Messaggio ");
        // se il messaggio non è stato inviato si forza l'invio
        if (res.isCommitted())
            res.reset();
        else
            res.flushBuffer();
}
```

Java Servlet

Tipologie di risposte:

- E’ possibile redirigere il client verso un URL differente impostando l’ header e il body appropriati:

  public void **sendRedirect**(String location) throws IOException

- Il parametro passato può essere un **URL relativo o assoluto**.
- In caso di **URL relativo** il server deve effettuare una trasformazione a URL assoluto. Nel caso in cui tale conversione non possa essere fatta per un qualsiasi motivo, allora verrà generata una eccezione di tipo **IllegalArgumentException**.

Java Servlet

Tipologie di risposte:

- L' invocazione del metodo **sendError**(), forza il completamento della trasmissione:
- Tale metodo permette di specificare un parametro che verrà utilizzato come messaggio di errore nell'header stesso.

- Con la versione 2.1 della API il metodo

  HttpServletResponse.**setStatus**(int sc, String sm)

  è stato deprecato a favore dei metodi

  HttpServletResponse.**setStatus**(int sc)

  HttpServletResponse.**sendError**(String msg)

  al fine di offrire maggiore eleganza e coerenza nella funzionalità dei metodi.

Java Servlet

Distruzione di un Servlet : metodo destroy()

- A completamento del ciclo di vita, si trova la fase di distruzione del Servlet, legata al metodo destroy(), il quale permette di :
  - Distruggere il Servlet dal container
  - terminazione del processo
  - log dello status
- La ridefinizione di tale metodo, derivato dalla interfaccia Servlet, permette di specificare tutte le operazioni complementari alla inizializzazione.

```
public void init(ServletConfig config) throws ServletException {
    // apertura della connessione verso il db
}
 public void destroy() {
// chiusura della connessione verso il db
}
```

Java Servlet

Distruzione di un Servlet : metodo destroy()

- Può accadere che al momento in cui il server invoca la **destroy**() il metodo **service**(). o altri metodi invocati, non abbiano terminato la loro attività
- Per evitare che questi metodi vengano brutalmente bloccate, occorre **gestire programmaticamente** l'attesa del completamento di tutte queste operazioni. Ciò richiede:
  - Gestione dei threads in esecuzione
  - Implementazione del metodo **destroy()**

Java Servlet

Gestione dei threads in esecuzione

```
public myServletShutdown extends HttpServlet {
    // variabile privata che memorizza il numero di thread attivi
    private int serviceCounter = 0;
    // metodo per l'incremento del numero di thread attivi
    protected synchronized void addRequest () {
        serviceCounter++;
    }
    // metodo per il decremento del numero di thread attivi
    protected synchronized void removeRequest () {
        serviceCounter--;
    }
    // metodo per ottenere il numero di thread attivi
   protected synchronized int getRequest() {
        return serviceCounter;
    }
}
```

Java Servlet

Implementazione del metodo **destroy()**

- Nel metodo **service**() verrà incrementato e decrementato il contatore

```
public void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException {
        addRequest();
        if (!isShuttingDown())
            metodoLungaEsecuzione();
        removeRequest();
    }
```

- Nel metodo **destroy**() viene gestito uno sleep di attesa

```
public void destroy() {
        // Attende lo stop di tutte le Request
        while(getRequest() > 0)
                Thread.sleep(interval);
}
```

Java Servlet

L'habitat di un Servlet: il Servlet Context

- Il **ServletContext** rappresenta l'ambiente di esecuzione all'interno del quale il Servlet viene fatto eseguire.
- L'interfaccia ServletContext mette a disposizione una serie di metodi piuttosto utili per
  - interagire con il contesto della applicazione
  - accedere alle risorse messe a disposizione del Servlet stesso.
  - effettuare il log di tutti gli eventi generati
  - modificare gli attributi ai quali il Servlet può accedere.

Java Servlet

L'habitat di un Servlet: il Servlet Context

- Il container è responsabile di:
  - **fornire una implementazione** della interfaccia ServletContext
  - permette l'utilizzo di una serie di **parametri di inizializzazione**, che verranno utilizzati dal Servlet all'interno del metodo init().

Java Servlet

L’habitat di un Servlet: il Servlet Context

- **getAttribute**() e **getAttribute**() permettono di impostare e ricavare un attributo dato il suo nome.

  public void setAttribute(String name, Object o)

  public Object getAttribute(String name)

- **getAttributeNames**() **removeAttribute**() consentono rispettivamente di ottenere la lista di tutti i riferimenti, o di rimuoverne uno dato il nome.

  public Enumeration getAttributeNames()

  public void removeAttribute(String name)

- **getContext**() permettere la comunicazione fra contesti differenti, che restituisce un ServletContext per un dato URL fornito come parametro.

  public static servletContext.getContext(String urlpath)

Java Servlet

La request delegation al RequestDispatcher

- E’ possibile inoltrare una richiesta ad un altro componente o Servlet dopo che ne abbia effettuata una prima elaborazione preliminare
- **getRequestDispatcher()** permette di inoltrare la request direttamente all’oggetto identificato dal path passato come parametro e torna un oggetto di tipo **RequestDispatcher**
- Questo oggetto prevede 2 metodi:

  public void **forward**(ServletRequest request, ServletResponse response) throws ServletException, IOException

  public void **include**(ServletRequest request, ServletResponse response) throws ServletException, IOException

Java Servlet

La request delegation al RequestDispatcher

- **forward()**
  - permette di inoltrare una richiesta pervenuta
  - al fine di garantire che il controllo passi totalmente al secondo oggetto, è necessario che l invocazione della forward() avvenga prima dell ottenimento di un **ServletOutputStream** o della creazione di un **Writer** da utilizzare per l invio dei dati verso il client.
- **include()**
  - il controllo resta al Servlet chiamante perciò la creazione dell'oggetto do output può essere fatta in ogni momento

Java Servlet

La request delegation al RequestDispatcher

```
//Viene ottenuto l’oggetto RequestDispatcher dal nostro contesto
RequestDispatcher dispatcher = getServletContext();

//si configura il contesto da richiamare e lo si invoca
dispatcher.getRequestDispatcher("/servlet/DBServlet?param=value");
dispatcher.include(req, res);

//si configura il contesto da richiamare configurandolo con l’oggetto req e lo si
    invoca
req.setAttribute("param", "value");
dispatcher.include(req, res);

//anche per il forward viene effettuata la stessa operazione
dispatcher.forward(req, res);
```

Java Servlet

Resource Abstraction

- Con la versione 2.1 della API è stato formalizzato il concetto di astrazione delle risorse, consentendo così l'accesso alle risorse di sistema indipendentemente dalla loro collocazione nel sistema.
- Questo permette di
  - gestire i Servlet come oggetti indipendenti dal contesto
  - possono essere spostati da un server a un altro
    - senza particolari difficoltà
    - senza la necessità di dover riscrivere buona parte del codice.
- Il metodo che permette di ottenere una risorsa è il

  ServletContext.getResource(String uripath)
- dove uripath rappresenta l URI in cui è collocata tale risorsa: è compito del web server effettuare il mapping fra risorsa vera e propria e URL.

Java Servlet

Scrittura su file di log

```
URL url = getServletContext().getResource("/myservlet.log");
URLConnection con = url.openConnection();
OutputStream out = con.getOutputStream();
PrintWriter pw = new PrintWriter(new OutputStreamWriter(out));
pw.println("Evento verificatosi il" + (new Date());
pw.close();
out.close();
```

Visualizzazione del contenuto di una pagina statica

```
URL url = getServletContext().getResource("/static_html/header.html");
out.println(url.getContent());
```

Java Servlet

Gestione del Logging

- **log(String msg, Throwable e)**
  - permette di scrivere su un file di log l'esito delle Exception che sono state riscontrate durante l'esecuzione delle Servlet.
- Presenta 2 parametri di ingresso:
  - **msg**: riporta il messaggio sotto forma testuale
  - **e**: riporta il messaggio sotto forma di stackflow

Java Servlet

Il context e la versione della API utilizzata

- L' evoluzione della Servlet API introduce problemi di compatibilità.
- Generalmente è garantita la retrocompatibilità ma può essere utile in certi casi ricavare la versione della API utilizzata
- Dalla 2.1 sono disponibili i seguenti metodi di ServletContext che restituiscono questo genere di informazione:

```
public int getMajorVersion()
public int getMinorVersion()
```

- Nel caso della versione 2.1 restituiscono rispettivamente i due interi 2 e 1.

Java Servlet

Invocazione della Servlet

- Su protocollo HTTP l'invocazione di una Servlet può essere effettuata utilizzando il metodo GET oppure POST
- La differenza fondamentale fra una GET ed una POST sta
  - nella modalità con cui viene effettuata l'invocazione
  - nella visibilità dei parametri
  - Caching delle invocazioni

Java Servlet

Modalità di invocazione:

- GET
  - occorre comporre l’URL della Servlet appendendo i parametri dopo il carattere ?.
  - la lunghezza massima di una invocazione di questo tipo deve essere di 256 caratteri.
- POST
  - parametri sono passati come elementi del BODY del pacchetto HTTP.
  - non si ha nessuna limitazione sulla lunghezza della invocazione,

Java Servlet

Visibilità dei parametri:

- GET
  - i parametri appaiono in chiaro nella URL di invocazione e non possono essere cifrati (protocollo SSL cifra il BODY)
- POST
  - I parametri non sono visibili in quanto vengono passati nel BODY HTTP (protocollo SSL può cifrare i dati contenuti)

Java Servlet

Caching delle richieste

- GET
  - la chiamata rimane in memoria nella cache del browser o può essere salvata nella memoria del web server al fine di ridurre il tempo di latenza da parte del client
- POST
  - non viene mai salvata in alcun tipo di memoria: è questa una precisa scelta progettuale e può essere sfruttata per aumentare il livello di sicurezza nel caso in cui, ad esempio, serva una estrema riservatezza sui dati.

Java Servlet

Cookies

- Poiché il protocollo HTTP e senza stato, non è possibile legare due transazioni separate in base alla sua specifica.
- Pertanto sono stati studiati i Cookies che consentono di memorizzare delle informazioni nella memoria del browser
- Il Servlet può creare un cookie nel browser del client ed aggiunge un campo nell’header HTTP . Durante la seconda transazione legge i valori memorizzati e ricostruisce l’esito della transazione precedente.
- In Java si possono creare e gestire i cookie tramite la classe javax.servlet.http.Cookie

```
Cookie C = new Cookie("uid","dedo12")
```

Java Servlet

Cookies

- Il nome di un cookie deve essere una stringa che non contenga nessuno dei caratteri speciali menzionati nella RFC 2068

  Netscape: [ ] ( ) = , " / ? @ : ;

- È possibile inoltre settare alcuni parametri
  - periodo massimo di vita, metodo setMaxAge() dopo il quale il cookie viene eliminato dalla cache del browser
  - un commento tramite il metodo setComment() che verrà presentato all'utente
- Una volta creato, per aggiungere il cookie al browser si può utilizzare il metodo

  HttpServletResponse.addCookie(C)

Java Servlet

Sessioni

- I **Cookies** consentono di memorizzare le informazioni sul **Client** e sono visibili anche da utenti diversi,
- Le **Sessioni** consentono di memorizzare le informazioni sul **Server** e sono accessibili solo durante la transazione.
- E' possibile utilizzare le Sessioni in 2 modalità:
  - Con l'ausilio dei **Cookies**
  - Attraverso **l'URL Rewriting**
- Nel **primo caso** viene memorizzato nel Cookies l'ID della transazione che viene recuperato dal Server per accedere alle informazioni della transazione
- Nel **secondo caso** viene riscritto l'URL che il client invocherà, in modo tale da accodare anche l'ID della transazione

Servlet e JDBC

JDBC

- La libreria JDBC (Java Database Connectivity) contiene un insieme di classi Java a disposizione dello sviluppatore che consente di utilizzare una serie di strumenti per l'interazione con un qualsiasi database server
- Un driver JDBC è un modulo software, dedicato ad uno specifico database, in grado di tradurre tutte le fuzionalità fornite dall'interfaccia JDBC in comandi del linguaggio di interrogazione adottato dal DataBase (nella maggior parte dei casi si tratta di SQL).

JDBC

- Mostriamo di seguito i passi necessari per interrogare un database e processare il risultato dell’interrogazione.
  1. Caricare il driver JDBC per il database interessato
  2. Apertura della connessione
  3. Preparare l’esecuzione della query
  4. Invio dell’interrogazione al database
  5. Ciclo sull’oggetto ResultSet per leggere i dati della query
  6. Chiusura della connessione

JDBC

1. Caricare il driver JDBC adatto al database da utilizzare
   - Il Class Loader Java si occupa di caricare in memoria le librerie necessarie per la connessione, non necessario in JDK1.6
- Si distinguono:
  - Per il Database Postgre:

    org.postgresql.Driver
  - Per il Database Oracle:

    oracle.jdbc.driver.OracleDriver
  - Per il Database Mysql:

    com.mysql.jdbc.Driver
  - Per il Database MicroSoft SQL Server :

    com.microsoft.jdbc.sqlserver.SQLServerDriver
- Esempio:

  Class.forName(“oracle.jdbc.driver.OracleDriver”);

JDBC

2. Apertura della connessione
   - Serve per creare la connessione impostando URL, user e pwd
- In cui
  - URL: è utilizzato per indicare i parametri di connessione al DB

    jdbc:TipoDiDatabase://IndirizzoDatabase:PortDatabase/SID
  - USER

    Utente con cui vogliamo connetterci, esistente nel DB
  - PASSWORD

    Password dell’utente
- Esempio:

```
Connection con = DriverManager.getConnection
    (“jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");
```

JDBC

3. Preparare l’esecuzione della query
   - Attivare la connessione per poter eseguire la query
- E’ possibile effettuare 3 tipi di operazioni:
  - createStatement : l’istruzione è inviata al DB al termine di tutte le operazioni
  - PrepareStatement: l’istruzione viene precarica in cache ed eseguita al termine delle operazioni
  - CallableProcedure: utilizzata per invocare Store Procedure
- Esempio:

```
Statement stat = con.createStatement();
```

JDBC

4. Invio dell’interrogazione al database
   - Eseguire la query ed ottenere il risultato in un oggetto ResultSet
- A seconda del tipo di operazione da effettuare possiamo avere:
  - executeQuery: per eseguire una Query
  - executeUpdate per eseguire un UPDATE
  - execute per eseguire una QUERY oppure un UPDATE
- Esempio:

  ResultSet res = stat.executeQuery(“ select * from cat”);

JDBC

5. Ciclo sull’oggetto ResultSet per leggere i dati della query
   - L’oggetto ResultSet contiene un cursore ai record della query
- Questo oggetto puo essere di grosse dimensioni se il risultato della query è molto lungo, pertanto occorre fare attenzione alla quantità di dati estratti in quanto possono occupare molta memoria
- Esempio:

```
while(res.next()) { out.println(res.getString(1) );
```

JDBC

6. Chiusura dello statement e della connessione
   - Terminata la query possiamo chudere la connessione al DB
- Nel momento in cui non abbiamo più la necessità di avere la connessione aperta per effettuare delle query, la possiamo chiudere.
- Altrimenti la teniamo aperta, in sessione, e la riutilizziamo successivamente
- Esempio:
  - res.close(); // chiusura ResultSet
  - stat.close(); //chiusura Statement
  - son.close(); //chiusura Connection

JDBC

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Query extends HttpServlet {
public void doGet(HttpServletRequest request,HttpServletResponse response) throws
     IOException, ServletException {
    String mat = request.getParameter("matricola");
    PrintWriter out = response.getWriter();
    Connection con = null;
    Statement stmt;
    ResultSet rs;
 // parametri di connessione
    String url = "jdbc:postgresql://SERVER-POSTGRESQL/esercitazioni";
    String user = "";
    String passwd = "";
```

JDBC

```
try {
        Class.forName("org.postgresql.Driver");
    } catch (ClassNotFoundException cnfe) {
        System.err.println("Driver JDBC non trovato: "+cnfe.getMessage());
    }
    out.println("<h1>Studenti</h1>");
    try {
        con = DriverManager.getConnection(url,user,passwd);
        String sql = " SELECT * FROM STUDENTE  WHERE matricola = '"+mat+"'";
        stmt = con.Statement();
        rs=stmt.executeQuery(sql);
        while (rs.next()) {
            out.println("<strong>Cognome:</strong> "+rs.getString("Cognome"));
            out.println("<strong>Nome:</strong> "+rs.getString("Nome"));
}
        con.close();
    } catch (SQLException sqle) {
         System.err.println("DriverManager non trovato: "+sqle.getMessage());
    } } }
```

Java Server Pages

Engine HTML

- Nel caso di CGI e Servlet, la pagina viene interamente costruita dal codice dell’applicazione, che scrive nello stream di output il codice HTML, riga per riga
- La creazione delle pagine HTML si può avvalere di:
  - librerie di funzioni o di oggetti che rappresentano i diversi tag HTML
  - Java Server Pages - in cui l’intero processo di generazione del codice, compilazione ed esecuzione è gestito automaticamente in modo trasparente e permette l’inserimento diretto di codice Java nella pagina.

JSP
Java Server Pages

- JSP è una pagina web il cui contenuto dinamico viene generato al momento in cui la pagina viene richiesta dal client.
- JSP è un file di testo scritto secondo le regole di un markup language in base al quale il contenuto del file viene elaborato da un JSP container, per restituire il risultato di una trasformazione del testo originale, secondo le istruzioni inserite nel testo.
- Una pagina JSP è un file in formato testo che comprende essenzialmente due tipi di testo:
  - "template text": ovvero testo “letterale” destinato a rimanere tale e quale dopo l’elaborazione della pagina;
  - "JSP text": porzioni di testo che vengono interpretate ed elaborate dal JSP container.

JSP
Java Server Pages

- JSP container converte la pagina JSP in un Servlet, generando prima il codice sorgente, poi compilandolo e successivamente eseguendolo al pari di una classe Java.

JSP
Java Server Pages

- Semplice esempio:

```
<html>
    <head><title>Data e ora</title></head>
    <body>
        <p>Data e ora corrente: <%= new java.util.Date() %></p>
    </body>
</html>
```

- Si tratta, come si vede, di una normale pagina HTML, con un solo elemento estraneo a questo linguaggio:

```
<%= new java.util.Date() %>
```

- E’ una espressione JSP che verrà interpretata e valutata dal JSP container e sostituita dalla data ed ora corrente.

JSP
Java Server Pages

- Il codice della Servlet prodotta dal Servlet Container potrebbe assomigliare a questo:

```
class JSPRawDate extends HttpJspBase {
    public void _jspService (HttpServletRequest request,
    HttpServletResponse response) {
PrintWriter out = response.getWriter();
out.println("<html>"); out.println();
out.println("<head><title>Data e ora</title></head>");out.println();
out.println("<body>");
out.println("<p>Data e ora corrente:” + new
    java.util.Date();out.println();
out.println("</body>");
out.println("</html>");
    }
}
```

JSP
Java Server Pages

- La classe è derivata da una ipotetica classe **HttpJspBase**, che sarà una sottoclasse di **HttpServlet** che dovrà implementare l'interfaccia **HttpJspPage** definita nella JSP API. Quest'interfaccia contiene il metodo **_jspService**(), corrispondente al metodo **service**() del GenericServlet..
- Il Servlet non fa altro che:
  - rimandare in output le parti del file che non contengono tag JSP come string literal
  - l'espressione JSP viene eseguita e valutata prima di essere scritta nello stream.

JSP
Java Server Pages

- Elementi di una pagina JSP:
  - Template text: testo / codice html
  - Scripting element
    - Comment: <%-- comment --%>;
    - Scriptlet <% code %>;
    - Declaration <%! declaration [declaration] ... %>;
    - Expression <%= expression %>;
    - Directive: <%@ directive ... %>;
  - Standard e Custom Action <jsp: .../>

JSP
Java Server Pages

- **Template text**:
  - Tutte le parti di testo che non sono definite come elementi JSP ma vengono copiate tali e quali nella pagina di risposta.
- **Comment**:
  - Con sintassi: <%-- comment --%>;
  - sono commenti che riguardano la pagina JSP in quanto tale, e pertanto vengono eliminati dal JSP container nella fase di traduzione–compilazione;
  - da non confondere con i commenti HTML e XML, che vengono inclusi nella risposta come normale template text.

JSP
Java Server Pages

- **Scriptlet**:
  - Con sintassi <% code %>; sono porzioni di codice che danno origine a porzioni di codice Java;
  - generalmente inserite nel metodo service() del Servlet;
  - se contengono dichiarazioni di variabili queste saranno variabili locali valide solo nell’ambito di una singola esecuzione del Servlet;
  - se il codice contiene istruzioni che scrivono sullo stream di output, il contenuto mandato in output sarà inserito nella pagina di risposta nella stessa posizione in cui si trova lo scriptlet.

JSP
Java Server Pages

- **Declaration**:
  - Con sintassi <%! declaration [declaration] ... %>;
  - sono dichiarazioni che vengono inserite nel Servlet come elementi della classe, al di fuori di qualunque metodo.
  - Possono essere sia variabili di classe che metodi. Se si tratta di variabili, la loro durata è quella del Servlet stesso; di conseguenza sopravvivono e conservano il loro valore nel corso di tutte le esecuzioni dello stesso oggetto Servlet.

JSP
Java Server Pages

- **Expression**:
  - Con sintassi <%= expression %>;
  - l’espressione viene valutata e scritta nella pagina di risposta nella posizione corrispondente a quella dell’espressione JSP.
- **Directive**
  - Con sintassi: <%@ directive ... %>;
  - sono direttive di carattere generale, indipendenti dal contenuto specifico della pagina, relative alla fase di traduzione–compilazione.

JSP
Java Server Pages

- Esempio di utilizzo di oggetti impliciti:

```
<html>
    <head><title>Data e ora</title></head>
    <%-- Inizio pagina jsp --%>
    <%@ page import = "java.util.*, java.text.*" %>
    <%! String DateFormat = "EEEE d MMMM yyyy“;
    DateFormat dateFormat = new SimpleDateFormat(DateFormat)
    <% Date dateTime = new Date(); %>
    <body>
        Oggi <%= dateFormat.format(dateTime) %>
    </body>
</html>
```

JSP
Java Server Pages

- Il primo elemento JSP è il **template text** che si presenta come codice HTML
- Il secondo elemento JSP è un **commento** della pagina
- Il terzo elemento JSP è una **direttiva** page di cui si specifica l'attributo import per importare dei package Java
- Poi c'è una JSP **dichiarazione** con cui si creano un oggetto di tipo DateFormat. Si usano le dichiarazioni JSP perché si desidera creare questi oggetti solo all'inizio e non a ogni esecuzione,
- Si usa uno **scriptlet** di una sola riga per creare un oggetto di tipo Date.
- L'ultimo elemento è una **espressione** JSP con la quala si visualizza la data.

JSP
Java Server Pages

- **Oggetti impliciti**
  - L’esecuzione della pagina JSP corrisponde all’esecuzione del metodo service() del Servlet.
  - Il Servlet manipola una serie di oggetti per svolgere il suo lavoro, principalmente i due oggetti ServletRequest e ServletResponse, su cui si basa tutto il meccanismo di funzionamento.
  - Per poter usufruire dei servizi del Servlet nella pagina JSP, occorre avere un modo per accedere a questi oggetti: a questo scopo esistono gli oggetti impliciti JSP, utilizzabili in tutti gli scriptlet.

JSP
Java Server Pages

```
public void _jspService ( HttpServletRequest request,
                          HttpServletResponse response)
throws java.io.IOException, ServletException {
    Object page = this;
    JspFactory _jspxFactory = JspFactory.getDefaultFactory();
    PageContext pageContext = _jspxFactory.getPageContext(this,
request, response,null, true, 8192, true);
    ServletContext application = pageContext.getServletContext();
    ServletConfig config = pageContext.getServletConfig();
    HttpSession session = pageContext.getSession();
    JspWriter out = pageContext.getOut();
  }
```

JSP
Java Server Pages

- Eccone un elenco.
  - request
  - response
  - out
  - page
  - pageContext
  - session
  - application
  - config
  - exception

JSP
Java Server Pages

- **request**
  - Corrisponde generalmente all’oggetto ServletRequest passato come parametro al metodo service() del Servlet;
  - può essere una qualunque sottoclasse di ServletRequest;
  - generalmente si tratta di una sottoclasse di HttpServletRequest.
- **response**
  - Corrisponde all’oggetto ServletResponse passato come parametro al metodo service() del Servlet;
  - può essere una qualunque sottoclasse di ServletResponse;
  - generalmente si tratta di una sottoclasse di HttpServletResponse.

JSP
Java Server Pages

- **out**
  - Poiché il container deve fornire un meccanismo di buffering, non è dato accesso direttamente all’oggetto PrintWriter restituito da response.getWriter().
  - L’oggetto out è invece uno stream bufferizzato di tipo javax.servlet.jsp.JspWriter, restituito da un metodo del PageContext.
  - Il trasferimento sullo output stream del ServletResponse avviene in un secondo tempo, dopo che tutti i dati sono stati scritti sull’oggetto out.

JSP
Java Server Pages

- **page**
  - È un riferimento all’oggetto che gestisce la richiesta corrente, cioè la Servlet.
  - In Java corrisponde al this dell’oggetto, pertanto è di scarsa utilità.

JSP
Java Server Pages

- **pageContext**
  - Si tratta di un oggetto della classe javax.servlet.jsp.PageContext utilizzata prevalentemente per incapsulare oggetti e features particolari di ciascuna implementazione del container.
  - Il PageContext contiene ad esempio un metodo che restituisce l’oggetto out, altri che restituiscono gli oggetti session, application, config e così via.
  - Il PageContext viene utilizzato anche per condividere oggetti tra diversi elementi

JSP
Java Server Pages

- **session**
  - Corrisponde all’oggetto HttpSession del Servlet, viene restituito da un metodo del PageContext.
- **application**
  - Corrisponde al ServletContext del Servlet, viene restituito da un metodo del PageContext.
- **config**
  - Corrisponde al ServletConfig del Servlet, viene restituito da un metodo del PageContext.
- **exception**
  - Quest’oggetto è disponibile solo all’interno di una error page

JSP
Java Server Pages

Esempio

```
<html>
<head><title>helloClient</title></head>
<body>
        <form action = “helloServer.jsp" method =
"post">
Scrivi qui il tuo nome
<input type = "text" name = "name">
<input type = "submit" value = "Clicca qui">
        </form>
</body>
</html>
```

```
<html>
        <head><title>helloServer</title></head>
        <body>
                <% String name = request.getParameter("name");
%>
                <% if (name == null || name.length() == 0) { %>
                Ciao, chiunque tu sia!
                <% } else { %>
                Ciao <%= name %>
                <% } %>
        </body>
</html>
```

JSP
Java Server Pages

- L’oggetto **request** viene utilizzato per prelevare il valore del parametro name, ricevuto tramite il post di un form HTML.
- Il **contenuto del messaggio** visualizzato dipende dal fatto che questo parametro esista e non sia una stringa vuota.
- Le **istruzioni condizionali** sono inserite in piccoli scriptlet Java;
- Come si vede, il codice può essere frammentato liberamente e **inframezzato** da altri elementi.

JSP
Java Server Pages

Standard e Custom Action:

- Come gli script, le action si traducono in istruzioni Java nel metodo **service**() del Servlet.
- Ma, a differenza degli script, seguono la **sintassi XML**
- **vantaggio** di essere piuttosto semplice, e quindi più facilmente alla portata di un web designer che spesso non ha competenze specifiche di programmazione.
- La specifiche JSP definiscono una serie di **tag standard** e un meccanismo per la definizione di nuovi tag (le cosiddette tag extension libraries)
- Le action, in quanto elementi XML, **possono essere** :
    - dotato di un **body**, con uno start-tag e un end-tag,
    - **empty**-element con un unico tag

JSP
Java Server Pages

Standard Action

- Le specifiche prevedono che le azioni standard siano implementate in tutti i container JSP, lasciando la possibilità di definire altri tags non standard per ciascuna implementazione.
- Tutti i tag delle azioni standard usano il namespace XML jsp, che è riservato e non può essere usato per le tag extension.
- Le azioni standard si possono dividere in due categorie:
  - action per l'uso di componenti JavaBeans;
  - altre action per compiere varie operazioni su pagine JSP

JSP
Java Server Pages

Componenti Java Beans

- I componenti JavaBeans sono probabilmente il mezzo migliore, più **pulito ed elegante**, per inserire contenuti dinamici in una pagina JSP.
- **Vantaggi**: massima separazione del codice dalla presentazione e di conseguenza la massima manutenibilità.

- I tag per la manipolazione di bean sono **tre**:
  - **jsp:useBean**: serve per utilizzare un bean già esistente o creare una nuova istanza di un bean.
  - **jsp:getProperty**: inserisce nella pagina il valore di una proprietà del bean.
  - **jsp:setProperty**: assegna il valore di una o più proprietà del bean

JSP
Java Server Pages

Separazione della logica di **View** dalla logica di **Business:**

```
<html>
   <head><title>Data e ora</title></head>
<body>
   <jsp:useBean id = "dateTime" class = "dates.DateTime" />
   Oggi è <jsp:getProperty name = "dateTime" property = "date"/>
</body>
</html>

import java.util.*;
import java.text.*;
public class DateTime {
   DateFormat dateFormat = new SimpleDateFormat("EEEE d MMMM yyyy");
   public String getDate() {
      Date date = new Date();
      return dateFormat.format(date);
   }
}
```

JSP
Java Server Pages

L’azione jsp:useBean:

- Questo tag serve per localizzare :
  - un bean già esistente
  - per creare un nuovo bean del tipo specificato.
- Gli attributi dell’action:
  - **Id** : identifica il bean nello scope specificato
  - **scope**: ambito di visibilità (page, request, session, application)
  - **class**: identifica la classe del bean
  - **type**: identifica il tipo di classe, anche se una super-classe
  - **beanName**: il bean viene creato per mezzo del metodo java.beans.Beans.instantiate()

JSP
Java Server Pages

Esempio

```
<jsp:useBean id = "bean" class = "SpecializedBean" type= "BaseBean" scope = "session" />
```

Nel metodo _jspService() del Servlet si troverà un codice di questo genere:

```
BaseBean bean = (BaseBean)pageContext.getAttribute("bean", PageContext.SESSION_SCOPE);
if (bean == null)
    bean = new SpecializedBean();
pageContext.setAttribute("bean", bean, PageContext.SESSION_SCOPE)
```

- Si controlla se l’oggetto è già presente nello scope specificato, per mezzo delmetodo PageContext.getAttribute().
- Se non è presente, viene creato e assegnato allo scope, con il metodo setAttribute().
- La variabile dichiarata è del tipo specificato dall’attributo type, ossia BaseBeanmentre mentre il tipo effettivo dell’oggetto creato è quello indicato dall’attributo class, cioè SpecializedBean.

JSP
Java Server Pages

- Quindi, all’interno di uno scriptlet, sarà possibile accedere ai bean anche attraverso gli oggetti impliciti, a seconda dello scope:

- con una chiamata del tipo

(BeanType)request.getAttribute("beanName“) per lo scope request;

- con una chiamata del tipo

(BeanType)session.getValue("beanName") per lo scope session;

- con una chiamata del tipo

(BeanType)application.getAttribute("beanName") per lo scope application;

- oppure si potrà usare il metodo

pageContext.getAttribute()

JSP
Java Server Pages

L’azione jsp:getProperty

- Con tale azione si può ottenere il valore di una proprietà dell’oggetto specificato. Il bean deve essere reso accessibile con una precedente azione jsp:UseBean.
- Gli attributi sono:
  - name: indica il nome del bean
  - Property: indica il nome della proprietà (regole JavaBean)

Esempio:

- <jsp:getProperty name = "bean" property = "email" />

JSP
Java Server Pages

L’azione jsp:setProperty

- Con tale azione si può settare il valore di una proprietà dell’oggetto specificato. Il bean deve essere reso accessibile con una precedente azione jsp:UseBean.
- Gli attributi sono:
  - name: indica il nome del bean
  - Property: indica il nome della proprietà (regole JavaBean)
  - Value: indica il valore da associare alla proprietà del Bean

Esempio:

- <jsp:setProperty name = "beanName" property = "propertyName" value = "explicitValue" />

JSP
Java Server Pages

L’azione jsp:include

- Con questa azione è possibile inserire il contenuto di una pagina statica o dinamica.
- A differenza della direttiva include, l’inclusione consentel ’inserimento di pagine dinamiche. Con le pagine dinamiche è possibile specificare dei parametri che verranno usati per la elaborazione della risposta.
- Questi sono gli attributi dell’azione:
    - Page: indica l’url della pagina
    - Flush: effettua il flush dell’output dopo l’inclusione

Esempio:

- <jsp:include page = "Hello.jsp" flush = "true">

JSP
Java Server Pages

L’azione jsp:forward

- Con jsp:forward possiamo inoltrare la richiesta in fase di elaborazione a un’altra pagina JSP o Servlet.
- Poiché l’oggetto request viene passato alla nuova pagina, i parametri e gli altri dati contenuti nell’oggetto saranno preservati nella nuova pagina.
- La risposta sarà totalmente a carico della nuova pagina.
- Questi sono gli attributi dell’azione:
    - Page: indica l’url della pagina

Esempio:

- <jsp:forward page = “Hello.jsp"/>

JSP
Java Server Pages

L’azione jsp:param

- Si usa con jsp:include o jsp:forward per inserire nuovi parametri nella richiesta che viene inoltrata alla pagina da includere o visualizzare separatamente. Si inserisce all’interno del body dell’azione principale.
- Questi gli attributi:
  - name: nome del parametro
  - Value: valore da assegnare

Esempio:

- <jsp:forward page = “Hello.jsp"/>

JSP
Java Server Pages

Direttiva: page

- La direttiva page contiene parametri che vengono applicati alla pagina corrente e a tutti i file inclusi con la direttiva include.
  - Non ha invece alcun effetto sui contenuti inclusi dinamicamente con jsp:include.
- La direttiva page può essere usata più volte nella pagina corrente
  - Ma ciascun attributo deve essere specificato una sola volta; quindi ogni direttiva deve contenere attributi differenti.
  - Fa eccezione l’attributo import, che può essere specificato più volte, con effetto cumulativo.
  - È consigliabile, per ragioni di chiarezza, inserire la direttiva page all’inizio della pagina.

JSP
Java Server Pages

Direttiva: page

- Gli attributi della direttiva page sono i seguenti:
  - language
  - extends
  - import
  - session
  - buffer
  - autoFlush
  - isThreadSafe
  - info
  - isErrorPage
  - contentType

JSP
Java Server Pages

Direttiva: page

- language
  - Indica il linguaggio di script usato negli script element.
  - Il valore di default è java.
  - I linguaggi utilizzabili, oltre a Java che deve essere supportato, dipendono dall’implementazione, quindi variano da container a container.

```
<%@ page language="java" %>
```

JSP
Java Server Pages

Direttiva: page

- extends
  - Specifica il nome completo (incluso il package) della classe di cui viene creata un’istanza per implementare in Java la pagina JSP.
  - Il default dipende dall’implementazione. La classe deve comunque implementare l’interfaccia JspPage.
  - Questo attributo si usa raramente,solo nel caso l’utente abbia implementato una propria classe per l’elaborazione delle richieste delle pagine JSP.

```
<%@ page extends=“NewHttpJspPage" %>
```

JSP
Java Server Pages

Direttiva: page

- Import
  - Una lista di packages di cui si vuole effettuare l’importazione per l’esecuzione degli script Java contenuti nella pagina.
  - La sintassi e il risultato sono gli stessi della import declaration del linguaggio Java, con la differenza che si possono specificare più package insieme, separati da virgola.
  - L’attributo import è l’unico che può essere usato più volte in una stessa pagina JSP.
  - Un’altra differenza è che in una pagina JSP viene effettuata l’importazione automatica non solo del package java.lang, ma anche di javax.servlet, javax.servlet.jsp e javax.servlet.http.

JSP
Java Server Pages

Direttiva: page

- session
  - Il valore può essere true(default) o false. Se il valore è true la pagina potrà utilizzare la sessione HTTP corrente dell’applicazione; se questa non esiste, viene creata.
  - La sessione può essere condivisa non solo con altre pagine JSP, ma anche con normali Servlet.
  - Se il valore è false l’oggetto session non sarà disponibile. Di conseguenza non si potrà neppure utilizzare un bean con scope = session.

```
<%@ page session="false" %>
```

JSP
Java Server Pages

Direttiva: page

- buffer
  - Indica se l’output stream deve essere bufferizzato
  - Il valore di default è 8kb.
  - Se si vuole usare uno stream non bufferizzato, si deve indicare il valore none. In questo caso verrà usato direttamente lo stream — restituito da getWriter() — dell’oggetto response.
  - Altrimenti ogni operazione di scrittura sarà diretta a un oggetto locale di tipo JspWriter, un particolare tipo di Writer che supporta l’opzione di autoflush.

```
<%@ page buffer="32kb" %>
```

JSP
Java Server Pages

Direttiva: page

- autoFlush
  - I valori possibili sono true e false. Il default è true.
  - Se il valore è true, quando il buffer è pieno, viene automaticamente effettuato un flush dello stream (in sostanza il contenuto del JspWriter viene inviato al ServletWriter dell’oggetto request).
  - Se il valore è false, verrà generata un’eccezione di overflow non appena si tenterà di scrivere oltre la dimensione del buffer. I valori possibili sono true e false. l default è true.

```
<%@ page buffer="5kb" autoFlush="false" %>
```

JSP
Java Server Pages

Direttiva: page

- isThreadSafe
  - I valori possibili sono true (default) e false.
  - Indica se la pagina è da considerarsi thread-safe.
  - Ogni volta che una pagina JSP viene richiesta, il container esegue il metodo _jspService() in processi distinti, quindi più istanze della stessa pagina potrebbero essere in esecuzione contemporaneamente. Ci sono situazioni particolari in cui più accessi simultanei ad una stessa risorsa potrebbero creare dei problemi
  - Se il valore è true, il JSP container potrà mandare più richieste concorrenti alla pagina JSP.
  - Se il valore è false, il container manderà le richieste una alla volta.

```
<%@ page isThreadSafe="false" %>
```

JSP
Java Server Pages

Direttiva: page

- info
  - Specifica una stringa di contenuto arbitrario inserita in qualità di Servlet info nella pagina compilata
  - E accessibile con il metodo getServletInfo() dell’oggetto page(this).

<%@ info="Esempio di direttiva page info" %>

- errorPage
  - Il valore specificato deve essere un relative URL.
  - Indica la pagina alla quale vengono mandate le eccezioni

  - <%@ errorPage="mioErrore.jsp" %>

JSP
Java Server Pages

Direttiva: page

- isErrorPage
  - Il valore può essere true o false (default).
  - Indica se la pagina corrente è una errorpage.
  - Se il valore è true, all’interno della pagina è disponibile l’oggetto implicito exception.

```
<%@ page  isErrorPage="true" %>
```

- contentType
  - Indica il MIME type e il tipo di codifica dei caratteri usata per la risposta.
  - Il valore di default è text/html; charset = ISO-8859-1.

```
<%@ page contentType="image/png" %>
```

JSP
Java Server Pages

Direttiva: include

- La direttiva include serve per inserire il contenuto di un file all’interno della pagina JSP.
- Come per le altre direttive, l’esecuzione viene effettuata a **translation time**, e il contenuto inserito è statico, ossia non può essere modificato con attributi della direttiva. Il che naturalmente non significa che il contenuto incluso non possa contenere elementi dinamici JSP
- La differenza rispetto all’azione jsp:include
  - eseguita in fase di processing (dal Servlet compilato) dà la possibilità di includere anche pagine “virtuali” ossia non esistenti nel file system, ma corrispondenti a indirizzi virtuali interpretati del web server (ad esempio un Servlet);
  - e possibile specificare dei parametri da inviare a pagine dinamiche. dell’applicazione.

JSP
Java Server Pages

Direttiva: include

- Attributi previsti per questa direttiva:
- file
  - L’URL del file che deve essere specificato come URL relativo, ossia o relativo alla locazione della pagina in cui compare la direttiva, oppure, con uno slash davanti, relativo alla root directory dell’applicazione.

JSP
Java Server Pages

Direttiva: taglib

- Questa direttiva permette di utilizzare action element non standard inclusi in una tag library, implementata secondo le specifiche JSP
- Gli attributi sono:
- uri
  - È lo Uniform Resource Identifier che identifica univocamente la tag library. Può essereun URL (Uniform Resource Locator) o un URN (Uniform Resource Name) come specificatodalle specifiche W3C (RTF 2396), oppure un pathname relativo.
- prefix
  - È il prefisso che identifica un namespace XML, usato per evitare conflitti tra nomi di tag di diversa origine. Il prefisso è obbligatorio. I prefissi jsp, jspx, java, javax,servlet, sun, sunw sono riservati alla Sun e non sono utilizzabili per custom tags.

JSP
Java Server Pages

Gestione degli errori

- La presenza di errori in una pagina JSP può essere ricondotta a diverse tipologie.
  - Errori in **fase di traduzione–compilazione**
    - Questi errori si traducono generalmente nell’invio da parte del server HTTP di un Server Error (errore 500).
    - La pagina dovrebbe riportare anche i messaggi generati dal JSP container, per errori di sintassi JSP, o dal compilatore Java per errori negli script element.

JSP
Java Server Pages

Gestione degli errori

- Errori in **fase di processing** (runtime errors)
  - Gli errori runtime si risolvono generalmente nella generazione di eccezioni Java.
  - Queste eccezioni, come per le normali classi Java, possono essere gestite
    - lasciate alla gestione di default del JSP container.
    - direttamente nella pagina tramite try/catch
    - Tramite uso delle error page che permette una gestione più ordinata e razionale.

JSP
Java Server Pages

Errore gestito dal container

```
<html>
<head><title>Hello</title></head>
<body>
<%
    String name = request.getParameter("name");
    if (name == null || name.length() == 0)
        throw new java.lang.Exception("Nome non inserito");
%>
Ciao <%= name %>
</body>
</html>
```

JSP
Java Server Pages

Errore gestito dal blocco try/catch

```
<html>
<head><title>Hello</title></head>
<body>
<% try {
   String name = request.getParameter("name");%>
   Ciao <%= name %>
<% } catch (java.lang.Exception e) { %>
   Nome non inserito
<% } %>
</body>
</html>
```

JSP
Java Server Pages

Errore gestito dall errorPage

```
<%@ page errorPage = "HelloError.jsp"%>
<html>
<head><title>Hello</title></head>
<body>
<%
   String name = request.getParameter("name");
   if (name == null || name.length() == 0)
       throw new java.lang.Exception("Nome non inserito");
%>
Ciao <%= name %>
</body>
</html>
```

```
<%@ page isErrorPage = "true" %>
<html>
<head><title>HelloError</title></head>
<body>
<%= exception.getMessage() %>
</body>
</html>
```

JSP
Java Server Pages

Elementi JSP con sintassi XML

- Ogni elemento JSP che non segua già la sintassi XML ha una sua corrispondente forma sintattica XML.
- Questo fa sì che una pagina JSP possa essere definita anche come un documento XML, permettendo così, tra l'altro, l'uso di tool basati su XML per la gestione di pagine JSP.

```
<%@ page ... %>                <jsp:directive.page ... />
<%@ taglib ... %>              attributo xmlns in jsp:root
<%@ include ... %>             <jsp:directive.include .../>
<%! ... %>                     <jsp:declaration>...</jsp:declaration>
<% ... %>                      <jsp:scriptlet>...</jsp:scriptlet>
<%= .... %>                    <jsp:expression> ... </jsp:expression>
attr = "<%= expr %>"           attr"%= expr %"
```

JSP
Java Server Pages

Configurazione e deployment

- Il deployment delle Java Server Pages segue le regole stabilite per le Web Application dalle specifiche dei Servlet di Sun.
- Tali specifiche comprendono i DTD (DocumentType Definition) dei Deployment Descriptor XML, che contengono informazioni sui componenti necessari al container per eseguire correttamente l'applicazione.
- Le specifiche forniscono inoltre delle raccomandazioni per
  - la struttura delle directory di una Web Application,
  - per il packaging in un file WAR

JSP
Java Server Pages

Il file web.xml

- Il file web.xml descrive i componenti dell'applicazione
- Non c'è nessuna particolare informazione da includere per le pagine JSP, a meno che
  - utilizzino tag extension library
  - siano incluse sotto forma di Servlet precompilati.
- In quest'ultimo caso, dopo aver compilato la pagina utilizzando i tools specifici del container, si devono includere le informazioni come per un normale Servlet, più un mapping sulla pagina JSP, come riportato in seguito.

JSP
Java Server Pages

Applicazione che utilizza una JSP come Servlet

```
<!DOCTYPE webappSYSTEM "http://java.sun.com/j2ee/dtds/web-app_1_2.dtd">
<webapp>
<servlet>
    <servlet-name> HelloWorld </servlet-name>
    <servlet-class> HelloWorld.class </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name> HelloWorld </servlet-name>
    <url-pattern> /HelloWorld.jsp </url-pattern>
</servlet-mapping>
</webapp>
```

JSP
Java Server Pages

Applicazione che usi una tag extension library:

```
<!DOCTYPE webappSYSTEM "http://java.sun.com/j2ee/dtds/web-app_1_2.dtd">
<webapp>
<taglib>
    <taglib-uri>http://www.miosito.it/examples/jsp/taglib</taglib-uri>
    <taglib-location>/WEB-INF/taglib/mb.tld</taglib-location>
</taglib>
</webapp>
```

JSP
Java Server Pages

Tag Library Descriptor

- Il Tag Library Descriptor è un file di configurazione in formato XML, che fornisce al container le informazioni necessarie per poter utilizzare correttamente la libreria.
- Elemento di root contenuto
  - taglib: definisce il tipo di documento, con attributi id e xmlns("http://java.sun.com/j2ee/dtds/jsptaglibrary_1_2.dtd").
  - Al suo interno vi sono tutta una serie di sottoelementi:

JSP
Java Server Pages

Sotto elementi obbligatori:

- tlibversion : indica il numero di versione della libreria.
- tag : descrive un tag della libreria. Deve esserne presente uno per ogni tag.
- shortname : un nome breve, di pochi caratteri, utilizzabile da un tool di authoring come prefisso.

Sotto elementi obbligatori:

- jspversion: indica la versione di JSP che deve essere supportata dal container. Il valore di default è 1.1.
- uri: un nome che identifica in modo univoco la specifica versione della libreria.
- info: un testo arbitrario contenente informazioni sulla libreria.

JSP
Java Server Pages

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "- //Sun Microsystems, Inc.
//DTD JSP Tag Library 1.1
//EN" "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>mio Tag</shortname>
  <uri>http://www.miosito.it/examples/jsp/taglib</uri>
  <info>Esempio di tag library</info>
  <tag>
...
</taglib>
```

JSP
Java Server Pages

I seguenti elementi sono sottoelementi dell’elemento tag, che descrivono le proprietà del tag,

Obbligatori:

- name : nome del tag.
- tagclass : classe Java usata per l’implementazione del tag.Deve implementare l’interfaccia javax.sevlet.jsp.tagext.Tag.
- attribute : definisce un attributo del tag. Se ne deve includere uno per ogni attributo.

JSP
Java Server Pages

Non obbligatori:

- teiclass: indica il nome della sottoclasse di TagExtraInfo eventualmente utilizzata per la definizione di scripting variables o per la validazione.
- bodycontent: indica il tipo di contenuto del body.
  - Può assumere uno dei seguentivalori:
    - JSP (default): il contenuto è elaborato secondo le regole delle pagine JSP.
    - empty: il body deve essere vuoto.
    - tagdependent: il body è elaborato dal tag handler;.
- info: un testo arbitrario contenente una descrizione del tag.

JSP
Java Server Pages

```
<taglib>
...
<tag>
    <name>Hello</name>
    <tagclass>HelloTag</tagclass>
    <teiclass>mbtags.IterateTagExtraInfo</teiclass>
    <bodycontent>JSP</bodycontent>
    <info>Primo esempio</info>
    <attribute>
...
</taglib>
```

JSP
Java Server Pages

I seguenti elementi sono sottoelementi dell'elemento attribute, che descrivono le proprietà dell'attributo:

- name (obbligatorio): il nome dell'attributo.
- required: indica se l'attributo è obbligatorio. Il valore di default è false.
- rtexprvalue: indica se l'attributo accetta request-time values, ossia una espressione JSP come valore. Il valore di default è false.

...

```
<attribute>
    <name>property</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
```

JSP
Java Server Pages

Classe che implementa la logica del Tag

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public HelloWorldTag implements Tag {
    PageContext pageContext =null;
    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().print("Hello World");
        } catch (Exception ex) {
            throw new JspException("IO problems");
        }
        return SKIP_BODY;
    }
}
```

JSP
Java Server Pages

TLD che definisce gli elementi del Tag

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.1//EN" "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>mt</shortname>
  <uri>http://www.mioSito.com/tutorials/mytags.jar</uri>
  <info>My first Tag library</info>
  <tag>
    <name>helloWorld</name>
    <tagclass>HelloWorldTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>A Hello World tag</info>
  </tag>
</taglib>
```

JSP
Java Server Pages

web.xml che indica i riferimenti al TLD

```
<taglib>
  <taglib-uri>mytags</taglib-uri>
  <taglib-location>/WEB-INF/taglib.tld</taglib-location>
</taglib>
```

Jsp page che importa il TLD e lo utilizza stampando il messaggio

```
<%@ taglib uri="mytags" prefix="mt" %>
<HTML>
  <HEAD>
    <TITLE>Hello World!</TITLE>
  </HEAD>
  <BODY><mt:helloWorld/></BODY>
</HTML>
```

Java Bean

Java Bean

- JavaBeans è una specifica, ossia un insieme di regole seguendo le quali è possibile realizzare in Java **componenti software** riutilizzabili, che abbiano la capacità di interagire con altri componenti, realizzati da altri produttori, attraverso un protocollo di comunicazione comune.
- Ogni Bean è caratterizzato
  - Dai servizi che è in grado di offrire
  - Un ambiente di sviluppo differente rispetto a quello in cui è stato realizzato.
- Un modello a componenti è caratterizzato da almeno sette fattori:
  - proprietà, metodi, introspezione, personalizzazione, persistenza, eventi e modalità di deployment.

Java Bean

- Proprietà
  - Le proprietà sono attributi privati, accessibili solamente attraverso appositi metodi get e set. Tali metodi costituiscono l'unica via di accesso pubblica alle proprietà,
- Metodi
  - I metodi di un Bean sono metodi pubblici Java. I metodi sono la prima e più importante via d'accesso ai servizi di un Bean.
- Introspezione
  - E' un processo che consiste principalmente nell'interrogare il componente per conoscerne i metodi, e dedurre da questi le caratteristiche.
- Persistenza
  - Un bean deve poter salvare il proprio stato per poter successivamente rispristinarlo (tramite la seralizzazione)

Java Bean

- Deployment
  - I bean possono essere incapsulati all’interno di un pacchetto jar ed essere deployati o venduti separatamente dall’applicazione.
- Eventi
  - La comunicazione del cambio di stato di un oggetto non viene lasciato alla gestione del programmatore ma segue un preciso meccanismo basato sugli eventi basato sul pattern Observer.

Java Bean

implementazione dei JavaBeans

- Le proprietà
  - sono attributi che descrivono l’aspetto e il comportamento di un Bean, e che possono essere modificate durante tutto il ciclo di vita del componente.
  - Tali proprietà possono essere di semplici, booleane o indicizzate

Java Bean

implementazione dei JavaBeans

- I metodi
  - Di base, le proprietà sono attributi privati, ai quali si accede attraverso una coppia di metodi detti accessor e mutator
- La coppia di metodi segue un “pattern design” a seconda che si tratti di:
  - Proprietà semplici
  - Proprietà booleane
  - Proprietà indicizzate

Java Bean

implementazione dei JavaBeans

- Qualora si tratti di **proprietà semplici**, si segue questa specifica:

```
public <PropertyType> get<PropertyName>()
public void set<PropertyName>(<PropertyType> property)
```

Esempio:

Definire per la proprietà foo di tipo Wombat i metodi:

```
public Wombat getFoo();
public void setFoo(Wombat w);
```

Java Bean

implementazione dei JavaBeans

- Qualora si tratti di **proprietà booleane**, si segue questa specifica:

```
public boolean is<PropertyName>();
public void set<PropertyName>(<PropertyType> property)
```

Esempio:

Definire per la proprietà foo di tipo Wombat i metodi:

```
public boolean isMarsupial();
public void setMarsupial(boolean m);
```

Java Bean

implementazione dei JavaBeans

- Qualora si tratti di **proprietà indicizzate**, si distinguono 2 casi:
  - get/set di **tutto l'array**
  - get/set di **parte dell'array**
- Nel **primo** caso si segue questa specifica:

```
public <PropertyElement>[ ] get<PropertyName>();
public void set<PropertyName>(<PropertyElement>[ ]);
```

Esempio:

Definire per la proprietà foo di tipo Wombat i metodi:

```
public Bah[ ] getFoo();
public void setFoo(Bah a[ ]);
```

Java Bean

implementazione dei JavaBeans

- Nel **secondo** caso si segue questa specifica:

```
public <PropertyElement> get<PropertyName>(int a);
public void set<PropertyName>(int a, <PropertyElement>
    b);
```

Esempio:

Definire per la proprietà foo di tipo Wombat i metodi:

```
public Bah getFoo(int a);
public void setFoo(int a, Bah b);
```

Java Bean

Le Serializzazione

- Per rendere serializzabile una classe Bean è di norma sufficiente implementare l'interfaccia Serializable
- L'interfaccia Serializable non contiene metodi: essa viene usata dal compilatore per marcare le classi che possono essere serializzate.
- Regole per implementare classi Serializable
  - dichiarare un costruttore privo di argomenti
  - definire al suo interno solamente attributi serializzabili altrimenti transient
- Per tutti i casi in cui la serializzazione standard non risultasse applicabile, occorre procedere all'implementazione dell'interfaccia Externalizable, fornendo, attraverso i metodi readExternal(ObjectInput in)e writeExternal(ObjectOutput out), delle istruzioni esplicite su come salvare lo stato di un oggetto su uno stream e come ripristinarlo in un secondo tempo.

Design Pattern

Design Pattern

- Un Design Pattern è una **soluzione assodata** (ossia convalidata dal suo utilizzo con successo in più di un progetto) ad un **problema ricorrente** in un contesto specifico (condizioni al contorno che vincolano la scelta della soluzione).
- Un pattern
  - Descrive un problema di progettazione comune
  - Descrive la soluzione al problema
  - Discute i risultati ed i vantaggi dell'utilizzo del pattern
- I pattern
  - Permettono il riutilizzo di tecniche di progettazione e architetture di successo
  - Conducono a sistemi più contenibili
  - Aumentano la produttività
  - Forniscono un linguaggio comune

Design Pattern

- I Design Pattern si distinguono in base allo scopo in:
  - Pattern di **creazione**: si applicano al processo di creazione degli oggetti
  - Pattern **strutturali**: trattano la composizione degli oggetti
  - Pattern **comportamentali**: modellano le modalità di interazione e le responsabilità tra le classi

Pattern di Creazione

- **Factory**: definisce una classe che crea una delle possibili sottoclassi di una classe astratta di base in funzione dei dati forniti
- Abstract Factory: fornisce un'interfaccia per creare famiglie di oggetti correlati senza specificare le loro classi concreti
- **Singleton**: assicura che una classe abbia una sola istanza e ne fornisce un punto globale di accesso
- Builder: consente di separare la procedura di costruzione di un oggetto complesso dalla sua rappresentazione, in modo che tale procedura possa creare più rappresentazioni differenti del medesimo oggetto

Pattern Strutturali

- **Adapter**: converte l'interfaccia di una classe in un'altra interfaccia che si attende un certo cliente; consente l'interazione tra classi che non potrebbero altrimenti cooperare in quanto hanno interfacce incompatibili
- Bridge: separa un'astrazione dalle sue implementazioni in modo da consentire ad entrambe di cambiare in modo indipendente
- Composite: compone oggetti in strutture ad albero che rappresentano gerarchie di tipo parte-per-il-tutto. Consente di trattare singoli oggetti e composizioni di oggetti
- **Flyweigth**: utilizza la condivisione per gestire in modo efficiente grandi numeri di oggetti
- Facade: Fornisce un'interaccia comune ad un insieme di interfacce in un sotto insieme. Facade definisce un'interfaccia di alto livello che rende un sottosistema più facile da utilizzare

Pattern Comportamentali

- Observer: definisce una dipendenza uno-a-molti tra oggetti in modo che quando un oggetto cambia stato, tutti gli oggetti che dipendono da esso ricevono una notifica e si aggiornano
- **Strategy**: definisce una famiglia di algoritmi, incapsula ciascuno di essi in una classe opportuna e li rende intercambiabili. Strategy fa si che gli algoritmi possano variare in modo indipendente dal codice che li utilizza.
- **Command**: incapsula le richieste di servizi in oggetti, consentendo di controllarne la selezione e la sequenza di attivazione, l'accodamento, l'annullamento
- Interpreter: dato un linguaggio, definisce una rappresentazione della sua grammatica e di un interprete che usa tale rappresentazione per valutare le frasi in quel linguaggio
- Iterator: fornisce un punto di accesso sequenziale agli elementi di un oggetto aggregato, senza rendere pubblica la sua organizzazione interna

Pattern Comportamentali

- **Template**: definisce lo scheletro di un algoritmo in un'operazione, delegando alcuni passi alle sotto classi; in pratica consente alle sotto classi di ridefinire alcuni passi di un algoritmo senza modificarne la struttura.
- Mediator: definisce un oggetto che incapsula la logica di interazione di un insieme di oggetti; favorisce il disaccoppiamento evitando agli oggetti di riferirsi esplicitamente l'un l'altro e consentendo così di variare in modo indipendente le loro interazioni.

Singleton

- Il Singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza. (http://it.wikipedia.org/wiki/Singleton)

```
public class MySingleton {
    private static MySingleton istanza = null;

    private MySingleton() {}

    public static MySingleton getMySingleton() {
        if (istanza == null)
            istanza = new MySingleton();
        return istanza;
    }
}
```

Factory

- Fornisce un metodo per istanziare un oggetto senza sapere a priori la sua esatta classe. Questo pattern raggiunge il suo scopo fornendo un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare. (http://it.wikipedia.org/wiki/Factory_method)

Factory

```
abstract class Pizza {
    public abstract double getPrice();
}
class PizzaMargherita extends Pizza {
    private double price = 8.5;
    public double getPrice() {
        return price;
    }
}
class PizzaMarinara extends Pizza {
    private double price = 10.5;
    public double getPrice() {
        return price;
    }
}
class PizzaCapricciosa extends Pizza {
    private double price = 11.5;
    public double getPrice() {
        return price;
    }
}
```

```
class PizzaFactory {
    public String [] pizzaType = {"Margherita","Marinara",
        "Capricciosa"};
    public static Pizza createPizza(String pizzaType) {
        if (pizzaType.equals("Margherita"))
            return new PizzaMargherita();
        else if (pizzaType.equals("Marinara"))
            return new PizzaMarinara();
        else if (pizzaType.equals("Capricciosa"))
            return new PizzaCapricciosa();
        else
            throw new IllegalArgumentException("La pizza " +
        pizzaType + " non esiste.");
    }
}
public class PizzaLover {
    public static void main (String avg[]) {

        System.out.println(PizzaFactory.createPizza("Margheri
        ta").getPrice());
    }
}
```

Adapter

- Il fine dell‘Adapter è di fornire una soluzione astratta al problema dell'interoperabilità tra interfacce differenti. Il problema si presenta ogni qual volta nel progetto di un software si debbano utilizzare librerie dotate di interfacce non perfettamente compatibili con quelle richieste da applicazioni già esistenti. Può essere comodo scrivere un Adapter che faccia da tramite tra le diverse interfacce, rendendole così compatibili. (http://it.wikipedia.org/wiki/Adapter)
- Possiamo distinguere 2 tipi di adapter:
  - Object Adapter: quando l’Adapter istanzia la classe da adattare
  - Class Adapter: quando l’Adapter implementa entrambe le interfacce da adattare

Object Adapter

Object Adapter

```
class RoundPeg {
    public void insertIntoHole(String str)
     {
       System.out.println("RoundPeg
     insertIntoHole(): " + str);
    }
}
class SquarePeg{
    public void insert(String str) {
       System.out.println("SquarePeg
     insert(): " + str);
    }
}
```

```
class PegAdapter extends SquarePeg {
    private RoundPeg roundPeg;
    public PegAdapter(RoundPeg peg)
      {this.roundPeg = peg;}
    public void insert(String str)
      {roundPeg.insertIntoHole(str);}
}
public class TestPegs {
    public static void main(String args[]) {

      SquarePeg squarePeg = new
     SquarePeg();
      squarePeg.insert("Inserting square
     peg...");

      PegAdapter adapter = new
     PegAdapter(new RoundPeg());
      adapter.insert("Inserting round peg...");
    }
}
```

Class Adapter

Class Adapter

```
interface RoundPegInterface {
   void insertIntoHole(String str);
}
interface SquarePegInterface {
   void insert(String str);
}
class RoundPeg implements RoundPegInterface {
   public void insertIntoHole(String str) {
      System.out.println("RoundPeg insertIntoHole(): " + str);
   }
}
class SquarePeg implements SquarePegInterface {
   public void insert(String str) {
      System.out.println("SquarePeg insert(): " + str);
   }
}
class PegAdapter implements RoundPegInterface, SquarePegInterface {
   private RoundPeg roundPeg;
   private SquarePeg squarePeg;
   public PegAdapter(RoundPeg peg) {this.roundPeg = peg;}
   public PegAdapter(SquarePeg peg) {this.squarePeg = peg;}
   public void insert(String str) {roundPeg.insertIntoHole(str);}
   public void insertIntoHole(String str){squarePeg.insert(str);}
}
```

```
public class TestPegs {
   public static void main(String args[]) {

      SquarePeg squarePeg = new SquarePeg();
      squarePeg.insert("Inserting square peg...");

      RoundPeg roundPeg = new RoundPeg();
      roundPeg.insertIntoHole("Inserting round peg...");

      SquarePegInterface roundToSquare = new
       PegAdapter(new RoundPeg());
      roundToSquare.insert("Inserting round peg...");

      RoundPegInterface squareToRound = new
       PegAdapter(new SquarePeg());
      squareToRound.insertIntoHole("Inserting square peg...");
   }
}
```

Flyweight

Flyweight è un Design pattern che permette di separare la parte variabile di una classe dalla parte che può essere riutilizzata, in modo tale da poter avere quest'ultima condivisa fra istanze differenti della parte variabile. (http://it.wikipedia.org/wiki/Flyweight_pattern)

Flyweight

```
abstract class Pizza {
    public abstract double getPrice();
}
class PizzaMargherita extends Pizza {
    private double price = 8.5;
    public double getPrice() {
        return price;
    }
}
class PizzaMarinara extends Pizza {
    private double price = 10.5;
    public double getPrice() {
        return price;
    }
}
class PizzaCapricciosa extends Pizza {
    private double price = 11.5;
    public double getPrice() {
        return price;
    }
}
```

```
class PizzaFactory {
    private static PizzaMargherita pizzaMargherita = new
        PizzaMargherita();
    private static PizzaMarinara pizzaMarinara = new
        PizzaMarinara();
    private static PizzaCapricciosa pizzaCapricciosa = new
        PizzaCapricciosa();
    public Pizza createPizzaMargherita() {
        return pizzaMargherita;
    }
    public Pizza createPizzaMarinara() {
        return pizzaMarinara;
    }
    public Pizza createPizzaCapricciosa() {
        return pizzaCapricciosa;
    }
}
public class PizzaLover {
    public static void main (String avg[]) {
        PizzaFactory pizzaFactory = new PizzaFactory();
        System.out.println((new
         PizzaFactory()).createPizzaMarinara().getPrice());
    }
}
```

Front Controller

Front Controller

- Questo Pattern prevede la presenza di un oggetto che si pone come controllore d’ingresso, per cui tutte le richieste devono essere gestite in modo centralizzato da un punto di accesso.
- Lo sviluppo di siti web prevede che l’utente venga indirizzato su una pagina di default che rappresenta il punto di ingresso all’applicazione.
- Questo consente di centralizzare i controlli in una pagina e non inserire il codice in tante pagine del nostro sito.

Front Controller

Value Object

Value Object

- Questo Pattern è uno tra i più usati nelle applicazioni Java.
- Si tratta di mappare dati con java bean.
- PROBLEMA: Un client effettua la richiesta di una risorsa remota e successivamente data la risorsa remota, richiede dettagli della risorsa stessa. Questo comporta un elevato traffico di risorse remote.
- Soluzione: i dettagli delle risorse vengono copiate in un oggetto temporaneo che viene restituito ed è disponibile localmente. Pertanto tutti i dettagli che devono essere richiesti, potranno essere richiesti senza generare traffico remoto ma solo locale (in quanto l’oggetto dei dettagli è locale).

Value Object

PROBLEMA

SOLUZIONE

Business Delegate

Business Delegate

- Viene utilizzato per disaccoppiare lo strato di "**presentazione** "con quello dei "**servizi di business**"
- Non occorre che lo strato di "**presentazione**" sia a conoscenza di dove siano locati i "**servizi di business**" e come fare per recuperarli, a questo penserà uno strato intermedio che è rappresentato dal "**business delegate**".
- Ciò riduce il numero di cambiamenti che occorre effettuare nella "**presentazione** "nel caso in cui cambi qualcosa nel "servizio di business". In questo caso solo il "**business delegate**" verrà modificato e non tutte le "**presentazioni**"

Business Delegate

Model View Controller

MVC

- MVC è un Pattern Architetturale che si basa sul Pattern Observer pertanto occorre prima trattare questo Design Pattern.
- Nel Pattern Observer esistono 2 attori principali:
  - **Soggetto**: colui che deve essere osservato
  - **Osservatore**: colui che osserva e che è interessato al cambiamento di stato del soggetto
- Nel momento in cui il soggetto cambia stato, notifica all'osservatore che il suo stato è cambiato ma non lo informa sui dettagli. Sarà l'osservatore che lo chiamerà per sapere esattamente cosa è avvenuto.
- Questo consente all'osservatore di essere sempre informato in tempo reale quando avverrà un cambiamento. Sarà però suo compito di richiedere informazione sui cambiamento intervenuti direttamente al soggetto interessato.

MVC

Stimolatore
stimola() {
soggetto.setStato(O
}
-soggetto
Soggetto
-stato
+registraOsservatore()
+rimuoviOsservatore()
+informaOsservatori()
+getStato()
+setStato()
-osservatori
<<interface>>
Osservatore
+aggiorna()
informaOsservatori(){
for all o in osservatori {
o.Aggiorna();
}
}
SoggettoConcreto
-stato
+getStato()
+setStato()
-soggetto
OsservatoreConcreto
-stato
+aggiorna()
aggiorna(){
stato = f(soggetto.getStato());
}

MVC

: Stimolatore
: SoggettoConcreto
o1 : OsservatoreConcreto
o2 : OsservatoreConcreto
1 : setStato()
2 : informaOsservatori()
3 : aggiorna()
4 : getStato()
5 : aggiorna()
6 : getStato()

MVC

- MVC è un Pattern Architetturale che viene utilizzato per disaccoppiare la:
  - **Presentazione (View)**: Contenuto presentato all’utente
  - **Comportamento(Controler):** Il comportamento dell’applicazione in base alle regole di business
  - **Dati(Model)**: I dati su cui si basano tali applicazioni
- Questo Pattern si basa sul Pattern Observer che viene impiegato per notificare il cambiamento di stato di un oggetto da osservare.

MVC

Presentazione

- Deve gestire la costruzione dell'interfaccia grafica (GUI) che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema.
- Per far sì che i dati presentati siano sempre aggiornati si adotta il pattern Observer, registrando la View come osservatore del Model.
- Pertanto qualora avvengono dei cambiamenti nello stato del Model, quest'ultimo notificherà la View che potrà richiedere gli aggiornamenti al Model in tempo reale.

MVC

Model

- Rappresenta i dati di business e la logica di business
- Incapsula lo stato dell’applicazione
- Risponde alle interrogazioni delle Viste
- Notifica alle view i cambiamenti di stato che richiedono una modifica della view stessa

MVC

Controller

- Ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model.
- Pertanto è un osservatore della View ed in caso di suo cambiamento di stato dovrà essere notificato dalla View
- Il Controller decide la sequenza di azioni necessarie per soddisfare la richiesta
- le azioni ammissibili sono rappresentate sotto forma di componenti Object Oriented, chiamati action class.

MVC

Modello
Rappresenta i dati
Rappresenta la logica business che agisce sui dati
Informa la view quando vi sono variazioni nello stato dei dati
Interroga lo stato del modello
Notifica i cambiamenti di stato
Cambia lo stato del modello
Vista
Visualizza i dati del modello
Supporta l'interazione con l'utente
Riceve gli aggiornamenti dal modello
Seleziona la view successiva per il client
Comunica la interazioni dell'utente al controllore
Controllore
Seleziona le view
Invia gli input dell'utente al modello (cambiamenti di stato)
Agisce come intermediario fra view e modello

MVC

Servlet container
Model
Controller
(servlet)
invoca
Actions
richiesta
HTTP
Client
(Browser)
web
server
invoca
risposta
HTTP
View
(page template)
legge
State
objects

MVC

Client
«Controller»
«Action»
Java Class
«Model Object»
Java Bean
«View Component»
JSP Template
doGet()
execute()
crea / usa
cede il controllo
legge da
Browser