CURS 4

Beans

• Orice clasă Java este un POJO (eng. Plain Old Java Object).

• JavaBeans: este o clasă Java specială. Reguli:

• Trebuie să aibă un constructor implicit (public și fără nici un parametru). Alte instrumente specializate vor folosi acest constructor pentru a instanția un obiect.

• Atributele trebuie să poată fi accesate folosind metode de tip getXyz, setXyz și isXyz (pentru atribute de tip boolean). Atributele pentru care sunt definite aceste metode se numesc proprietăți, numele proprietății fiind xyz

• Clasa trebuie sa fie serializabilă => permite instrumentelor specializate să salveze și să refacă starea

Spring

= framework open-source creat pentru a ajuta la dezvoltarea aplicațiilor complexe și foarte mari

• În Spring se pot folosi obiecte simple Java (POJO)

• Spring promovează cuplarea slaba prin “injectarea” asocierilor(/dependentelor prin pincipiul IoC) și folosirea interfețelor. IMPORTANT

IoC, Dependecy Injection

• = Inversion of Control sau dependency injection (DI).

•= procesul prin care obiectele își definesc asocierile (dependențele) fie prin parametrii constructorilor, fie prin argumentele unei metode de tip factory sau prin proprietăți de tip set apelate imediat după crearea obiectului

• Un container “injectează” aceste dependențe când creează obiectul

• În Spring, obiectele care formează un sistem soft sunt gestionate de containerul bazat pe IoC și sunt numite bean-uri.

• Un bean Spring este un obiect Java obișnuit care este instanțiat, asamblat și gestionat de containerul Spring IoC.

• Bean-urile și asocierile dintre ele sunt descrise în datele de configurare

# Bean-uri inner

• nu necesită specificarea atributului id

• Aceste bean-uri nu pot fi refolosite. Sunt folosite pentru “injectare” o singură dată și nu pot fi referite de alte bean-uri

CURS 5  
Șablonul Proxy

• Asigură pentru un obiect existent, un surogat sau un înlocuitor în scopul controlării accesului la acesta

•Proxy la distanță (remote proxy) – pentru un obiect în alt spațiu de adresă,

•Proxy virtual (virtual proxy) – pentru un obiect mare din memorie,

•Proxy de protecție - controlează accesul la obiectul original

# Șablonul Remote Proxy

• oferă un înlocuitor local pentru un obiect aflat în alt spațiu de adresă/memorie.

• Înlocuitorul este responsabil cu codificarea unei cereri, a parametrilor și trimiterea lor către obiectul real aflat într-un spațiu de adresă diferit.

• Clientul cererii crede că comunică cu obiectul real, dar este un proxy între ei.

• Proxy-ul transformă cererile clientului în cereri la distanță, obține rezultatul cererii și îl transmite clientului.

Șablonul Data Transfer Object (DTO)

• Un DTO este un obiect care conține informația ce trebuie transmisă între unul sau mai multe procese pentru a reduce numărul de apeluri

• Fiecare apel de metodă remote este costisitor

• Un alt obiect este responsabil cu conversia datelor din model într-un DTO și invers

•Un DTO conține, de obicei, multe atribute și metode de tip get/set pentru acestea

• Un DTO ar trebui folosit ori de câte ori este necesară transmiterea mai multor date între două procese într-un singur apel de metodă.

CURS 7

Remote Procedure Call

• Apelul procedurilor la distanță (RPC) = tehnologie de comunicare între procese care permite unei aplicații să inițieze execuția unei subrutine sau a unei proceduri în alt spațiu de adrese, fără ca programatorul să scrie explicit codul corespunzător interacțiunii dintre procese. Adica, programatorul scrie aproximativ același cod indiferent dacă apelează o rutină locală sau una la distanță

• folosit pentru implementarea aplicațiilor client-server.

• Un apel la distanță este inițiat de client prin trimiterea unei cereri către un server la distanță pentru execuția unei proceduri cu parametrii dați. Serverul trimite un răspuns clientului, iar aplicația își continuă execuția.

• Cât timp serverul procesează cererea clientului, execuția clientului este blocată

•Avantaje

•Dezvoltarea ușoară a aplicațiilor client-server

•Ascunderea detaliilor de implementare pentru comunicarea client-server

•Dezavantaje

•Aplicațiile se pot dezvolta într-un singur limbaj

•Dacă apar versiuni noi pentru interfețe (servicii), trebuie modificați toți clienții

# gRPC

•Cu gRPC o aplicație client poate apela metode la distanță ca si cum ar fi metode ale unui obiect local.

• Facilitează crearea aplicațiilor distribuite și a serviciilor.

• folosește idea definirii unui serviciu (interfețe) care specifică metodele ce pot fi apelate la distanță, împreuna cu parametrii și tipul returnat al acestora

• În aplicația server există un obiect remote care implementează interfața, iar serverul gRPC gestionează cererile clienților

•Clientul are un proxy care oferă aceleași metode ca și obiectul remote

• Implicit, gRPC folosește Protocol Buffers ca și limbaj de definire a tipurilor de mesaje și a serviciilor (IDL)

# Interfețe IDL

•Interface Definition Language (IDL) permite specificarea interfețelor obiectelor remote

• Interfața unui obiect remote indică metodele ce pot fi apelate la distanță; nu cum sunt implementate aceste metode

•Clienții depind doar de interfață

• Interfețele IDL nu depind de un anumit limbaj de programare

Protocol Buffers (Protobuf)

•=modalitate independentă de limbaj și platformă de a serializa structuri de date

•flexible, eficiente și au un mecanism automat de serializare a datelor structurate

• Se definește o singura dată modul de structurare a datelor; se folosește un compilator special (protoc) care generează cod ce permite scrierea/citirea structurilor de date în/din o varietate de streamuri de date și folosind diferite limbaje de programare.

• Este permisă modificarea structurii datelor fără a provoca apariția erorilor în programele dezvoltate folosind vechea structură de date.

• Specificarea structurii datelor se definește folosind tipuri de mesaje protocol buffers și sunt salvate în fișiere .proto.

Object/Relational Mapping (ORM)

=tehnică de programare pentru convertirea informațiilor dintr-un sistem orientat-obiect într-o bază de date relațională

•Principiul ORM este de a delega altor instrumente managementul persistenței și de a lucra doar cu entitățile din domeniu, nu cu structurile dintr-o bază de date relațională

• Instrumentele de mapare obiect-relație stabilesc o legătură bidirecțională între o bază de date relațională și obiectele din system

• Mapare=Determinarea modului în care obiectele și relațiile dintre ele vor fi păstrate într-un mediu de stocare permanent

• Maparea proprietății=> descrie cum va fi stocată valoarea proprietății

• Maparea relațiilor=> descrie cum vor fi persistate relațiile dintre unul sau mai multe obiecte (asociere, agregare, moștenire).

## Shadow Information

=informații pe care obiectele trebuie să le păstreze (pe lângă informațiile normale) pentru a putea fi persistate(ex Cheia primară)

ORM Impedance Mismatch

• Paradigma orientată obiect promovează dezvoltarea aplicațiilor folosind obiecte care păstrează date, dar conțin și logica aplicației

• Bazele de date relaționale stochează datele în tabele și manipulează datele folosind proceduri stocate și interogări SQL.

• Diferențele dintre cele două abordări au fost numite: object-relational impedance mismatch

# Strategii pentru Impedance Mismatch

### •Maparea moștenirii

### •Maparea relațiilor dintre obiecte

### •Maparea proprietăților statice

## Maparea moștenirii

•Bazele de date relaționale nu suportă moștenirea.

•Programatorul trebuie să mapeze moștenirea dintre entitățile din modelul orientat obiect într-o bază de date relațională

• Tehnici:

• Maparea ierarhiei de clase într-o singură tabelă.

•Reprezentarea unei ierarhii de clase ca și o singură tabelă cu coloane pentru toate atributele din toate clasele din ierarhie.

• Fiecare clasă păstrează informațiile relevante pentru ea într-o înregistrare din tabelă. Coloanele care nu sunt relevante rămân goale.

•Când se încarcă un obiect din tabelă, instrumentul ORM trebuie să știe ce clasă să instanțieze.

• În tabelă se adaugă o coloană care indică ce clasă ar trebui instanțiată

• Avantaje: Există doar o singură tabelă în baza de date si nu e nevoie de operații join

• Dezavantaje: Nu toate câmpurile din tabelă sunt relevante, e confuza. poate deveni prea mare

• Maparea fiecărei clase într-o tabelă

• Fiecare clasă din ierarhie are tabela ei

Inregistrarile se leaga fie prin folosirea cheii primare atât în tabela corespunzătoare clasei de bază cât și în clasa derivată, fie Fiecare tabelă să aibă cheia primară proprie, și folosirea cheii străine pentru a păstra legătura cu tabela corespunzătoare clasei de bază

• Interogările asupra bazei de date sunt dificile

Avantaje: Toate coloanele sunt relevante pentru fiecare înregistrare, tabelele sunt mai ușor de înțeles și nu se folosește spațiu în mod ineficient

Dezavantaje: Este necesară folosirea mai multor tabele pentru a încărca un obiect din mediu persistent; Orice refactorizare cauzează modificarea structurii bazei de date; Tabelele claselor de bază cauzeaza probleme de performanță din cauza accesării dese

• Maparea fiecărei clase concrete în tabela ei.

• Fiecare clasă concretă (non-abstract) din ierarhie are tabela ei.

• Fiecare tabelă conține coloane pentru toate proprietățile din ierarhie până la ea. Atributele din clasa de bază sunt duplicate în tabelele corespunzătoare subclaselor

• Avantaje: Fiecare tabelă păstrează toate informațiile relevante și nu are câmpuri irelevante; Nu este nevoie de operații join pentru citirea datelor; Fiecare tabelă este accesată doar când clasa respectivă este accesată

• Dezavantaje: Dacă câmpurile din modelul obiectual sunt mutate în ierarhie, trebuie modificate definițiile tabelelor; Dacă se modifică un câmp dintr-o clasă de bază, trebuie modificate toate tabelele corespunzătoare subclaselor; O operație de căutare folosind clasa de bază, necesită căutari în toate tabelele

• Maparea claselor într-o structura de tabele generică

Șablonul Identity =Cheia primară dintr-o bază de date relațională este păstrată printre atributele obiectului pentru a păstra legătura dintre un obiect în memorie și o înregistrare din baza de date

Maparea cheii străine => Mapează o asociere dintre obiecte ca și cheie străină între tabelele dintr-o bază de date relațională

=>poate fi folosită pentru aproape toate asocierile dintre clase. Nu poate fi folosită pentru n-la-n.

## Maparea folosind o tabelă de asociere

=> Tabela are doar două coloane corespunzătoare cheilor străine, conține câte o înregistrare pentru fiecare pereche de obiecte asociate. • Tabela de legătura nu are echivalentul unui obiect în memorie (nu are ID). Cheia primară este compusă din cheile primare ale celor două tabele asociate.

## Maparea proprietăților statice – Strategii:

• O tabelă cu o singură înregistrare, o singură coloană pentru fiecare proprietate statică • Pro: Simplu, acces rapid • Con: multe tabele mici

• O tabelă cu mai multe coloane, o singură înregistrare pentru fiecare clasă• Pro: Simplu, acces rapid • Con: multe tabele mici, dar mai puține decât la strategia precedentă

• O singură tabelă cu mai multe coloane - o singură înregistrare pentru toate clasele• Pro: număr minim de tabele introdus • Con: potențiale probleme de concurență dacă mai multe clase trebuie să acceseze datele în același timp.

CURS 10

# Servicii Web

=sisteme de schimbare a mesajelor bazate pe XML care folosesc Internetul pentru interacțiunea dintre aplicații

=o colecție de protocoale și standarde folosite pentru transmiterea datelor între aplicații sau sisteme

• Nu este dependent de un anumit limbaj de programare sau de un anumit sistem de operare

• structura de baza: XML + HTTP

REST

• REpresentational State Transfer=stil arhitectural pentru sisteme distribuite hypermedia

6 constrângeri: 1. Client–server 2. stateless 3. Cacheable 4. uniform interface 5. layered system 6. code on demand -opțională

• Conceptul de bază în REST este resursa(=orice informatie)

• datele și funcționalitățile sunt considerate resurse ce pot fi accesate folosind Uniform Resource Identifiers (URIs)

• Clienții și serverele interschimbă (schimbă între ele) reprezentări ale resurselor folosind o interfață standardizată și un protocol standardizat (adesea HTTP)

• REST este un stil arhitectural ce permite proiectarea unor aplicații slab cuplate (loosely coupled ) folosind HTTP

Uniform interface• Pentru a obține o interfață uniformă, se folosesc mai multe constrângeri arhitecturale pentru a ghida execuția componentelor.

• REST este definit de 4 constrângeri ale interfeței:

• Identificarea resurselor

• Manipularea resurselor folosind reprezentările

• Mesaje self-descriptive

• Hypermedia ca și motor al stării aplicației.

• Dezvoltatorii trebuie să decidă interfața pentru resursele sistemului făcute publice clienților și să folosească întotdeauna această interfață. • O resursă din sistem trebuie să aibă un singur URI

REST - Client–server=> Aplicațiile client și server trebuie să poate evolua independent fără dependențe între ele

REST – Stateless=> Toate interacțiunile client-server trebuie să fie stateless. • Serverul nu va păstra informații despre ultima cerere HTTP de la client. • Fiecare cerere va fi tratată ca și cum ar fi o cerere nouă• Fiecare cerere trimisă de la client la server trebuie să conțină toate informațiile necesare pentru a înțelege cererea• Starea sesiunii este păstrată în întregime la client. •Constrângerea REST stateless: la fiecare cerere HTTP trebuie transmise informațiile necesare

Cacheable cere ca datele dintr-un răspuns la o cerere să fie implicit sau explicit marcate ca cacheable sau non-cacheable• Caching trebuie aplicat tuturor resurselor ce se declară cacheable.

Code on demand(opțională)=> Serverul poate trimite și cod executabil, pentru a suporta o parte a aplicației

CURS 12

Comunicare sincronă

• Tehnologiile RPC ca gRPC și Thrift folosesc comunicarea sincronă.

• Când un client apelează o metodă a unui obiect remote, clientul trebuie să aștepte terminarea execuției înainte de a trece mai departe, chiar daca metoda nu returneaza nimic

• Clientul este cuplat cu serviciul prin interfața serviciului. Dacă interfața serviciului se modifică, toți clienții acestuia trebuie să se modifice corespunzător.

• Clientul este dependent de locația serviciului

• Clientul este dependent de disponibilitatea serviciului. Dacă serviciul devine indisponibil, clientul va eșua.

Comunicarea asincronă

•= modalitate de a trasmite indirect mesaje de la o aplicație la altă aplicație, fără a aștepta un răspuns.

• Când mesajele sunt trimise asincron, clientul nu trebuie să aștepte terminarea procesării mesajului sau livrarea mesajului.

• Clientul trimite mesajul și trece la execuția următoarei instrucțiuni, presupunând că serviciul va primi si va prelucra la un moment dat mesajul trimis

• Există doi actori principali: brokerii de mesaje și destinațiile.

• Când o aplicație trimite un mesaj, îl trimite unui broker de mesaje.

• Brokerul de mesaje asigură livrarea mesajului la destinația specificată, permițând expeditorului continuarea execuției.

• Mesajele trimise asincron au specificată locația de unde mesajele pot fi preluate(=destinatia)

• Există două tipuri de bază de destinații: cozi și topic-uri.

• point-to-point (corespunzător cozilor)

• publish/subscribe (corespunzător topicurilor).

# Point-To-Point Model

• => fiecare mesaj are exact un expeditor și un destinatar. Când brokerul de mesaje primește un mesaj, pune mesajul într-o coadă. Când destinatarul cere următorul mesaj, mesajul este scos din coadă și trimis acestuia.

• Deoarece mesajul este șters din coadă în momentul trimiterii, se garantează livrarea acestuia unui singur destinatar.

• Chiar dacă fiecare mesaj din coada de mesaje este livrat unui singur destinatar, nu înseamnă că doar un singur destinatar scoate mesaje din coadă.

# Publish-Subscribe Model

• In modelul publish/subscribe, mesajele sunt trimise unui topic. Asemănător cozilor, mai mulți destinatari pot aștepta livrarea unor mesaje de la un topic. Spre deosebire de cozi, când un mesaj este trimis unui topic toți destinatarii care așteaptă vor primi o copie a mesajului respectiv.

• Expeditorul (publisher) nu știe care sunt destinatarii (subscribers). El știe doar că mesajul va fi trimis unui topic, nu știe cine așteaptă mesaje la topicul respectiv.

• Expeditorul nu știe nici cum vor fi procesate mesajele.

## Avantajele comunicării asincrone

• Nu există timpi de așteptare

• mesajele trimise asincron sunt centrate pe date; Clientul nu trebuie să știe particularitățile specifice serviciului

• Independența locației;clienții sistemelor bazate pe mesaje nu știu ce serviciu va procesa mesajele lor sau locația acestora; Dacă la un moment dat un serviciu devine suprasolicitat, se pot porni noi instanțe ale aceluiași serviciu care se abonează la aceeași coadă.

• Livrare garantată; Dacă mesajele sunt trimise asincron, clientul are garanția că mesajele sale vor fi livrate. Chiar dacă un serviciu nu este disponibil în momentul trimiterii mesajului, mesajul va fi păstrat de brokerul de mesaje până când serviciul va deveni din nou disponibil

# Java Message Service (JMS)

• =Un standard Java care definește o interfață comună pentru lucrul cu brokeri de mesaje

• Folosind JMS, toți brokerii de mesaje pot fi accesați folosind aceeași interfață comună

## Advanced Message Queuing Protocol (AMQP)

• În JMS există 3 tipuri de participanți:

• expeditorul mesajului

• destinatarul

• destinația=>decuplează expeditorul de destinatar, dar amândoi sunt cuplați de destinație; pentru:

• primirea mesajelor

• determinarea modului în care mesajele vor fi distribuite destinatarilor: coadă/ topic

• Producătorii AMQP nu trimit mesajele direct unei cozi

• AMQP introduce un nou nivel între producător și cozile care păstrează mesajele: exchange

• Un producător de mesaje trimite mesajul unui exchange; Exchange-ul rutează mesajele cozilor corespunzătoare; Consumatorii iau mesajele din cozi și le procesează; Producătorii trimit mesajele unui exchange cu o cheie de rutare; consumatorii iau mesajele din coada

## RabbitMQ

RabbitMQ este un broker de mesaje bazat pe AMQP;

• Ideea de bază a modelului RabbitMQ este că producătorul NU trimite mesajul direct unei cozi; Producătorul nu știe nici dacă mesajul chiar va fi distribuit unei cozi; Un exchange primește mesajele de la producători și le trimite spre cozile corespunzătoare

Producător=>un program care trimite mesaje

Coadă=>o zonă tampon mare de mesaje; Mai mulți producători pot trimite mesaje aceleiași cozi. Mai mulți consumatori pot citi/primi mesaje de la aceeași coadă

Consumator=>un program care așteaptă să primească mesaje.

• Fanout exchange - distribuie mesajele tuturor cozilor cunoscute.

• Direct exchange: - mesajul va fi distribuit cozilor care au asociate ca și chei exact cheia de rutare a mesajului

• topic exchange nu pot avea o cheie de rutare arbitrară. • Cheia de rutare trebuie să fie o listă de cuvinte separate prin punct

CURS 13

•WebSocket-urile sunt o conexiune persistentă, bi-direcțională, full-duplex de la un browser web la un server

•După ce conexiunea a fost stabilită, ea rămâne deschisă până când clientul sau serverul decide să o închidă

• Evenimentele din ciclul de viață:

• Un client inițiază conexiunea trimițând o cerere de tip HTTP handshake.

• Serverul răspunde cu un handshake.

• Conexiunea este stabilită. Din acest moment conexiunea este simetrică, bidirecțională.

• Ambii participanți (clientul și serverul) pot să trimită și să primească mesaje.

• Unul dintre participanți închide conexiunea

JSON Web Token (JWT)

• JSON Web Token este un standard care definește un mod compact și independent de a transmite informații între participanți ca și un object JSON.

• Compact: are dimensiuni reduce

• Independent: un token JWT conține toată informația necesară despre o entitate pentru a evita interogări multiple ale bazei de date.

• Informația conținută într-un token JWT poate fi verificată și se poate avea încredere în ea pentru că este semnată digital

Structura unui token JWT • 3 părți separate prin ‘.’: aaa.bbb.ccc • Header (aaa) • Payload (bbb) • Semnătura digitală (ccc)

Header JWT • Conține informații despre tipul de token și algoritmii de criptare folosiți pentru conținut. • Are două părți: • tipul tokenului (JWT) • algoritmul de criptare

Payload JWT • Conține informații despre entitate ce pot fi verificate (identitatea utilizatorului, permisiunile, etc), eng. Claims

Semnătura digitală JWT • Este folosită pentru a verifica expeditorul și pentru a verifica dacă mesajul a fost modificat pe parcurs