UNIVERZITET METROPOLITAN BEOGRAD

CS230 - DISTRIBUIRANI SISTEMI

Implementacija poslovnog nivoa pomoću zrna sesija

Lekcija 08

PRIRUČNIK ZA STUDENTE

CS230 - DISTRIBUIRANI SISTEMI

Lekcija 08

IMPLEMENTACIJA POSLOVNOG NIVOA POMOĆU ZRNA SESIJA

UVOD

Enterpirise JavaBeans dozvoljavaju programerima da se lakše fokusiraju na razvoj poslovne logike.

Većina Java EE aplikacija mora da vodi računa o uobičajenim zahtevima kao što su transakcije, bezbednost, skalabilnost i tako dalje. EJB (Enterpirise JavaBeans) dozvoljavaju programerima da se fokusiraju na razvoj poslovne logike bez potrebe za vođenjem računa o implementaciji navedenih zahteva. Postoje dva tipa EJB zrna:

- zrna sesije (session beans);
- zrna vođena porukama (message-driven beans).

U ovom delu lekcije će akcenat biti na analizi i demonstraciji zrna sesije koja u velikoj meri pojednostavljuju razvoj programske logike na serverskoj strani. Vodeći se navedenim, akcenat u lekciji će biti na sledećim temama:

- Predstavljanje zrna sesije;
- Kreiranje zrna sesije;
- EJB upravljanje transakcijama;
- Primena aspektno - orijentisanog programiranja sa presretačima;
- EJB Timer servis;
- Generisanje zrna sesije iz JPA entiteta.

U novim verzijama Java EE odbačen je koncept zrna entiteta u korist JPA.

Savladavanjem ove lekcije student će u potpunosti razumeti i vladati primenom zrna sesija u Java EE aplikacijama.

Poglavlje 1

Kreiranje zrna sesije

UVOD U ZRNA SESIJE

Zrno sesije enkapsulira poslovnu logiku za Java EE aplikaciju

Zrno sesije enkapsulira poslovnu logiku za Java EE aplikaciju. Nove verzije Java platforme za razvoj kompleksnih i distribuiranih aplikacija, uvođenjem zrna sesije značajno su olakšale i unapredile razvoj budući da se sva pažnja programera sada fokusira na razvoj poslovne logike umesto da se rasipa vođenjem računa o zahtevima aplikacije kao što su bezbednost, skalabilnost, transakcije i tako dalje.

> Iako se zahtevi poput transakcija i bezbednosti ne implementiraju direktno , moguće ih je podešavati primenom anotacija.

Postoje tri tipa trna sesije:

- zrno sesije bez stanja (stateless session beans);
- zrno sesije sa stanjem (stateful session beans);
- singlton zrno sesije (singleton session beans).

Zrno sa stanjem održava stanje konverzacije sa klijentima između poziva metoda, a zrno bez stanja nema tu sposobnost. U aplikaciji može da postoji samo jedna instanca singlton zrna, dok je moguće da aplikativni server kreira više instanci zrna sa i bez stanja.

KREIRANJE ENTERPRISE PROJEKTA

Postoje različiti tipovi Java projekta u kojima mogu biti kreirana zrna sesije.

Postoje različiti tipovi Java projekta u kojima mogu biti kreirana zrna sesije:

- *Enterprise Application*,
- *EJB Module*,
- *Web Application*.

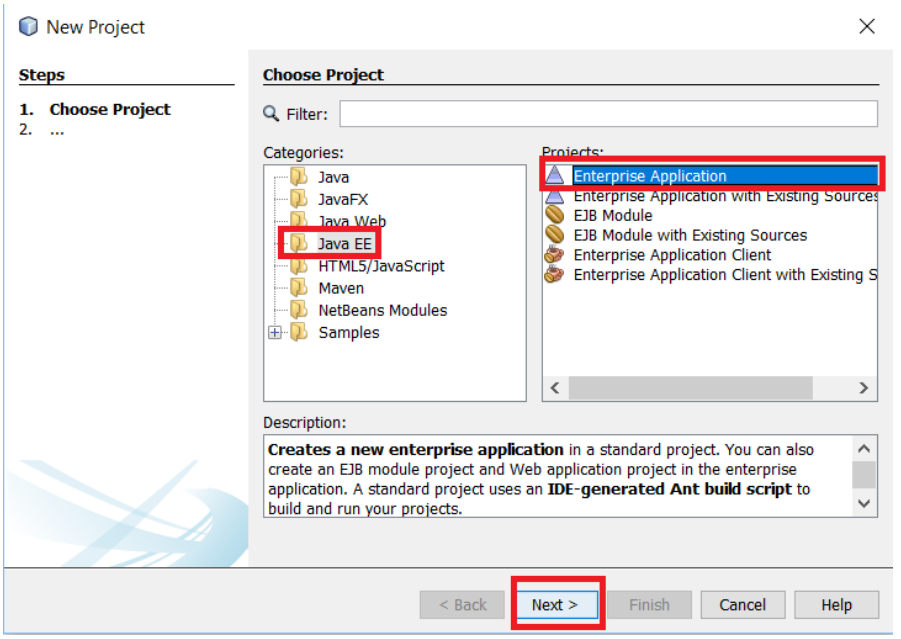
EJB Module projekat sadrži damo EJB zrna dok *Enterprise Application* projekat sadrži EJB zrna zajedno sa njihovim klijentima, u formi Java Web aplikacije ili "standalone" Java aplikacije.

Mogućnost dodavanja EJB zrna u veb aplikaciju postoji od verzije Java EE 6. Sada postoji mogućnost jednostavnijeg pakovanja i angažovanja veb aplikacija primenom EJB zrna. Kod veb aplikacije i kod EJB zrna je sada moguće čuvati u jedinstvenoj WAR datoteci (Web Archive).

Prilikom angažovanja Java EE aplikacije kroz aplikativni server GlassFish, moguće je uposliti samostalne klijente kao deo aplikacije. Ovi nezavisni klijenti su nakon toga dostupni kroz *Java Web Start* (http://www.oracle.com/technetwork/java/javase/javawebstart/index.html). Ovaj alat dozvoljava jednostavno pristupanje EJB zrnima iz klijentovog koda primenom anotacija. "Pravi" nezavisni klijenti, koji se izvršavaju izvan aplikativnog servera, zahtevaju poštovanje *Java Naming and Directory Interface* (JNDI) za obezbeđivanje reference na EJB zrno.

Za lakše razumevanje problematike biće, kao i u ostalim lekcijama, kreiran adekvatan primer. U ovom slučaju biće kreirani zrno sesije i Java Web Start klijent koji će funkcionisati u okviru iste Java EE aplikacije.

Za početak, u razvojnom okruženju NetBeans, bira se File | NewProject, načon čega se bira tip aplikacije *Enterprise Application* iz kategorije *Java EE*. Navedeno je prikazano sledećom slikom.

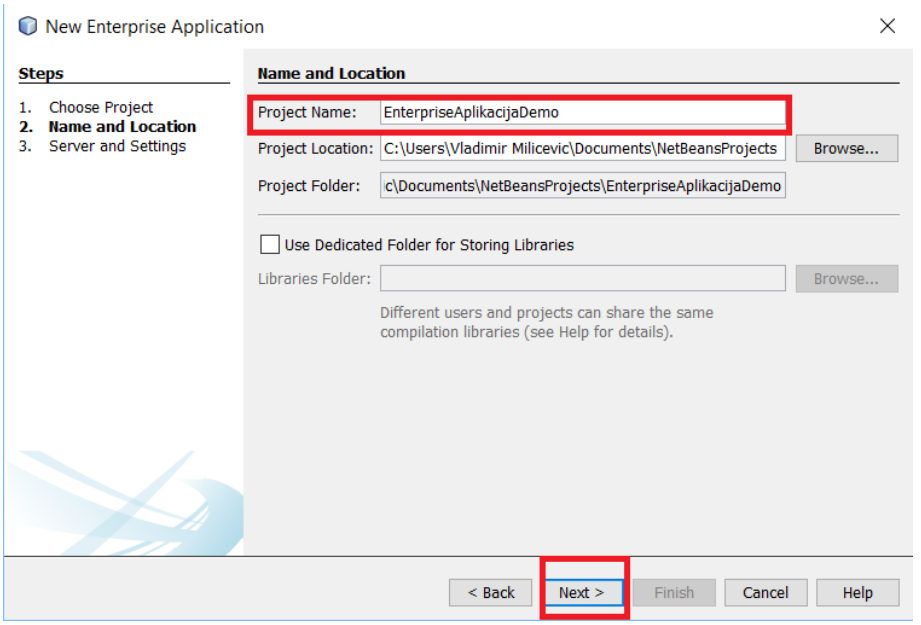


Slika 1.1 Kreiranje Java EE projekta [izvor: autor]

KREIRANJE PROJEKTA ZRNA SESIJE

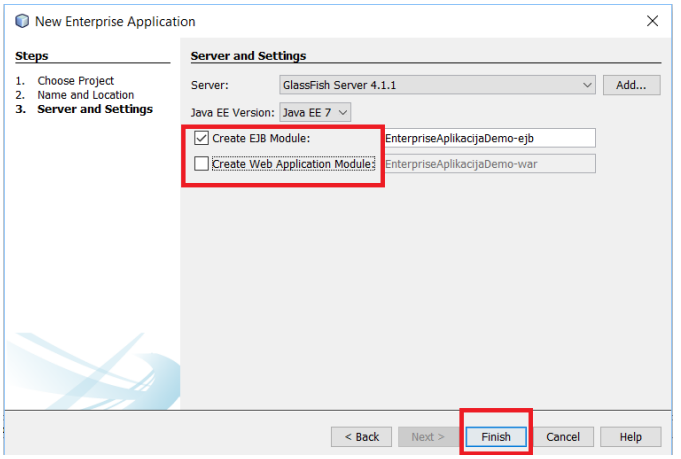
U nastavku definiše se naziv projekta i kreira zrno sesije.

Klikom na dugme Next u prozoru sa Slike 1, otvara se novi prozor u kojem se definiše naziv projekta. To je prikazano sledećom slikom.



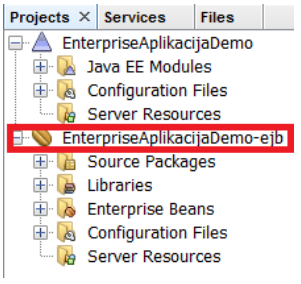
Slika 1.2 Definisanje naziva Java EE aplikacije [izvor: autor]

Klikom na dugme Next (Slika 2) otvoriće se novi prozor u kojem je neophodno izabrati module koji će biti uključeni u Java EE aplikaciju. Predlaže se izbor opcije *Create EJB Module*, dok bi opcija *Create Web Application Module* u ovom trenutku trebalo da bude isključena. Navedeno je prikazano sledećom slikom.



Slika 1.3 Izbor opcije za kreiranje zrna sesije [izvor: autor]

Klikom na dugme Finish (Slika 3) završeno je kreiranje Java EE projekta koji sadrži EJB zrno pod nazivom *EnterpriseAplikacijaDemo-ejb*. Navedeno je ilustrovano sledećom slikom.

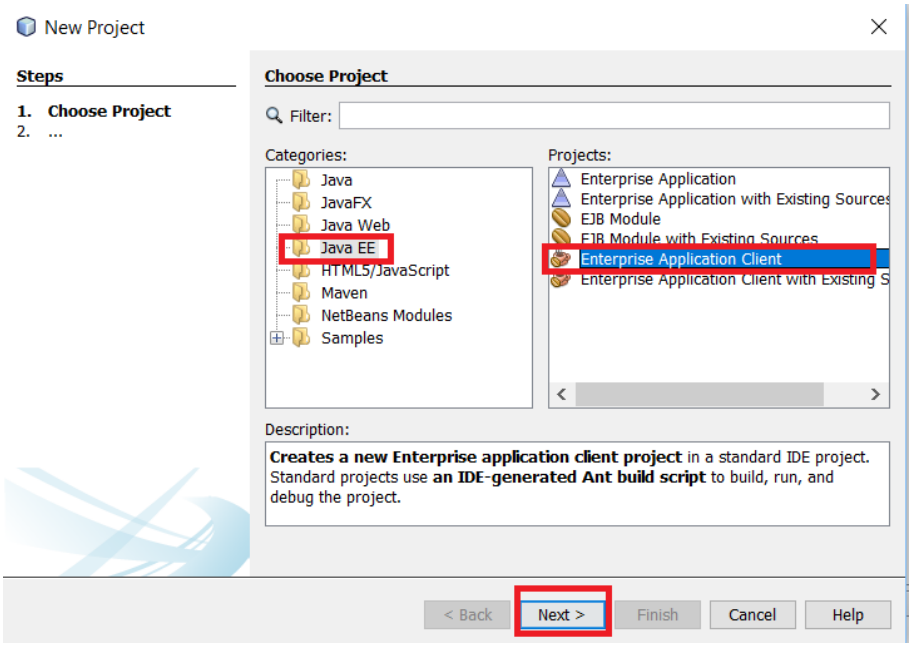


Slika 1.4 EJB zrno u kreiranom projektu [izvor: autor]

PROJEKAT KLIJENTA JAVA EE APLIKACIJE

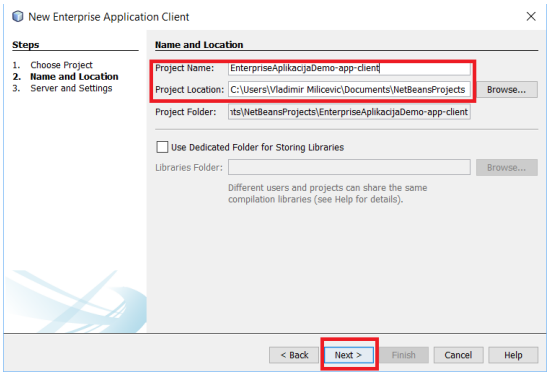
Projekat će sadržati i datoteku tipa Enterprise Application Client.

Pre kreiranja bilo kakve programske logike, neophodno je dovršiti definiciju strukture primera. Kao što je već napomenuto, pored EJB zrna, primer će sadržati i projekat tipa *Enterprise Application Client* iz kategorije *Java EE,* čarobnjaka *New Project.* Navedeno je priloženo sledećom slikom.



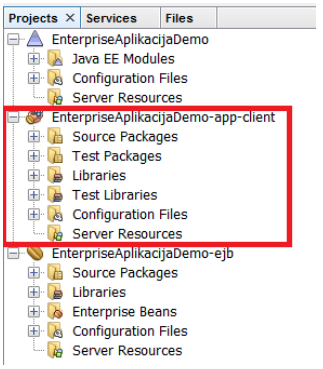
Slika 1.5 Kreiranje projekta Enterprise Application Client [izvor: autor]

Kao i za sve ostale projekte neophodno je obaviti završna podešavanja koja podrazumevaju dodelu naziva projektu, lokaciju i tako dalje.



Slika 1.6 Podešavanje projekta [izvor: autor]

Klikom na Next otvara se novi prozor koji zahteva potvrdu prethodno urađenih zadataka i izbor aplikativnog servera klikom na dugme Finish. Primer je sada spreman za razvoj programske logike.

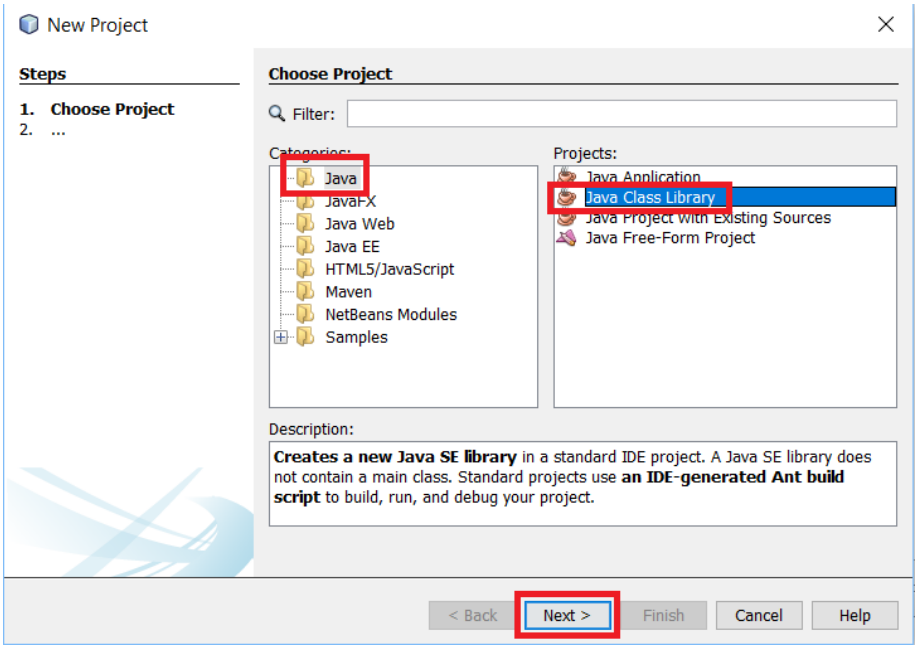


Slika 1.7 Kreirani projekat klijenta [izvor: autor]

KREIRANJE JAVA BIBLIOTEKE KLASA

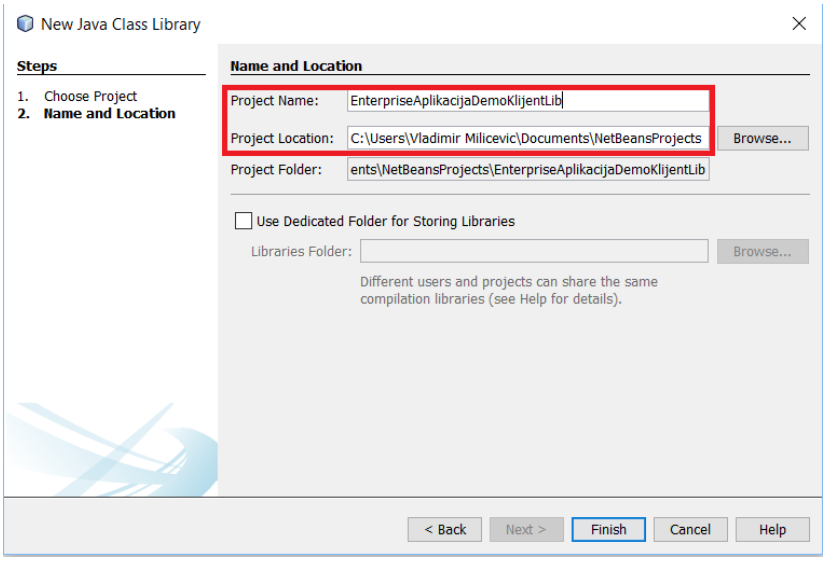
Neophodno je kreirati Java biblioteku klasa za udaljeni interfejs zrna sesije.

Budući da će EJB klijent biti distribuiran na različitim Java virtuelnim mašinama, nego što je slučaj sa kreiranim EJB zrnom, neophodno je kreirati Java biblioteku klasa za udaljeni interfejs zrna sesije. U tu svrhu, moguće je kreirati novi *Java Class Library* projekat iz kategorije *Java* primenom razvojnog okruženja NetBeans IDE (sledeća slika).



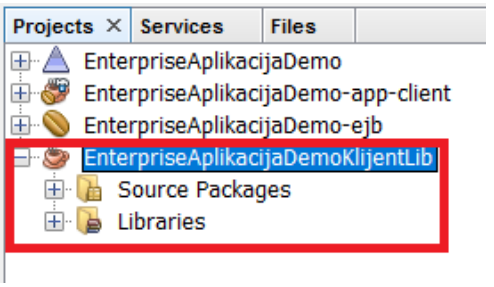
Slika 1.8 Kreiranje projekta Java biblioteke klasa [izvor: autor]

Takođe, neophodno je projektu dodeliti odgovarajući naziv i lokaciju, kao što je prikazano sledećom slikom.



Slika 1.9 Dodatna podešavanja projekta Java biblioteke klasa [izvor: autor]

Klikom na Finish, kreiran je i ovaj projekat i integrisan u prateći primer.



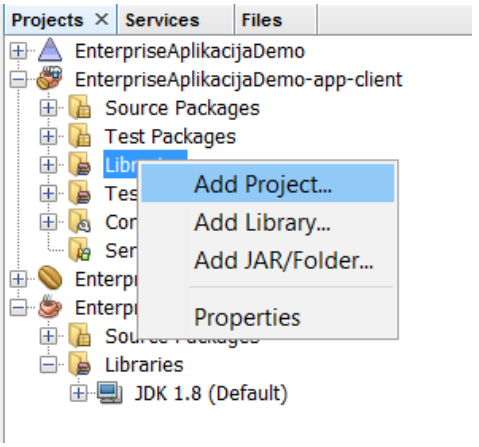
Slika 1.10 Projekat Java biblioteke klasa [izvor: autor]

POVEZIVANJE JAVA BIBLIOTEKE KLASA SA DISTRIBUIRANIM KLIJENTIMA

Kreirani projekat tipa Java Class Library je neophodno snabdeti odgovarajućim bibliotekama.

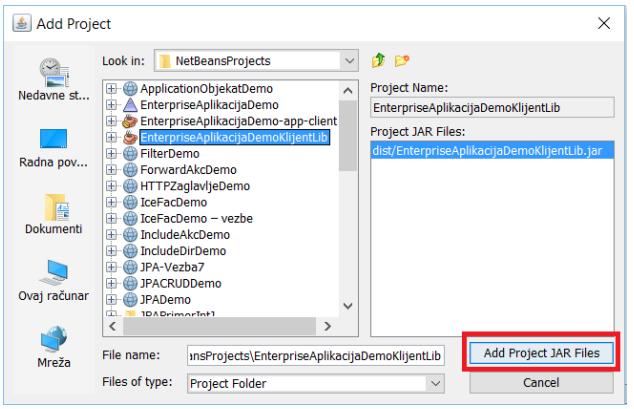
U prethodnom delu lekcije je pokazano kako se kreira Java biblioteka klasa za web distribuiranu Java EE aplikaciju. U nastavku je neophodno pokazati kako se *Java biblioteka klasa* povezuje sa konkretnim projektom, u ovom slučaju sa projektom klijenata EJB zrna.

U projektu klijenta EJB zrna, koji se u primeru naziva *EnterpriseAplikacijaDemo-app-client,* desnim klikom na folder *Libraries,* otvara se prozor (meni) u kojem je neophodno izabrati opciju *Add Project.* Navedeno je prikazano sledećom slikom.



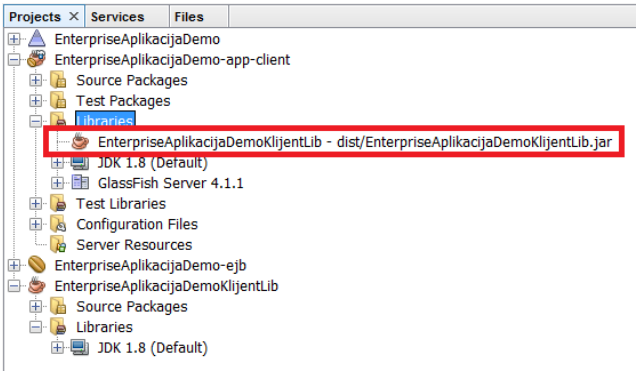
Slika 1.11 Dodavanje biblioteke klasa u projekat klijenata EJB zrna [izvor: autor]

U nastavku je neophodno izabrati kreiranu Java biblioteku klasa. Klikom na opciju *Add Project* otvara se prozor u kojem se bira kreirana Java biblioteka klasa, a to je pokazano sledećom slikom.



Slika 1.12 Izbor Java biblioteke klasa [izvor: autor]

Klikom na dugme *Add Project JAR Files* završava se podešavanje primera i za EJB klijente povezana je Java biblioteka klasa (sledeća slika).

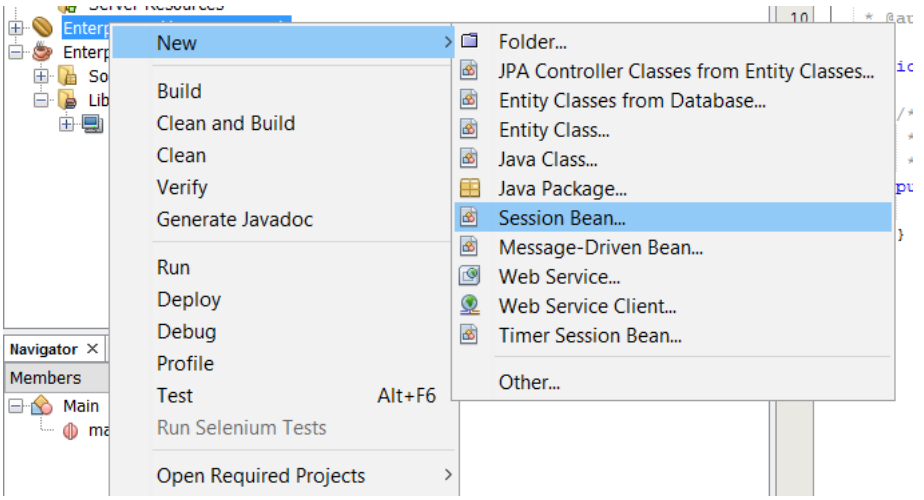


Slika 1.13 Povezana Java biblioteka klasa u projektu EJB klijenata [izvor: autor]

KREIRANJE ZRNA SESIJE

Sa okončanim podešavanjima moguće je pristupiti kreiranju zrna sesije.

U prethodnom izlaganju lekcija se bavila podešavanjem projekta, odnosno, ispunjavanjem uslova za kreiranje prvog *zrna sesije*. Budući da su svi uslovi ispunjeni moguće je pristupiti ovom zadatku. Zadatka će biti obavljen nad kreiranim EJB modulom, koji se u ovom konkretnom slučaju naziva *EnterpriseAplikacijaDemo-ejb.* Prvo se obavlja desnim klik na EJB modul, a potom izborom New | Other bira se tip datoteke *Session Bean.* Navedeno je ilustrovano sledećom slikom.



Slika 1.14 Kreiranje nove datoteke zrna sesije [izvor: autor]

Izborom tipa datoteke *Session Bean* otvara se novi prozor koji forsira izvršavanje nekoliko različitih zadataka:

- Definisanje naziva *zrna sesije* koje se kreira;

- Specificiranje paketa za zrno sesije;
- Izbor tipa zrna sesije: *zrno sa stanjem*, *zrno bez stanja* ili *singlton zrno;*
- Izbor lokalnog ili udaljenog interfejsa za klijente EJB zrna.

Zrno sa stanjem je korisno ako se želi sačuvati vrednost bilo koje varijable klijenata koji su u stanju konverzacije (između poziva metoda). Zrno bez stanja ne održava stanje konverzacije ali se brže izvršava. Singletone postoji od Java EE 6 i prilikom svakog angažovanja aplikacije postoji tačno jedna singlton instanca. Poslednja zrna su veoma korisna kod keširanja frekventnog čitanja iz baza podataka.

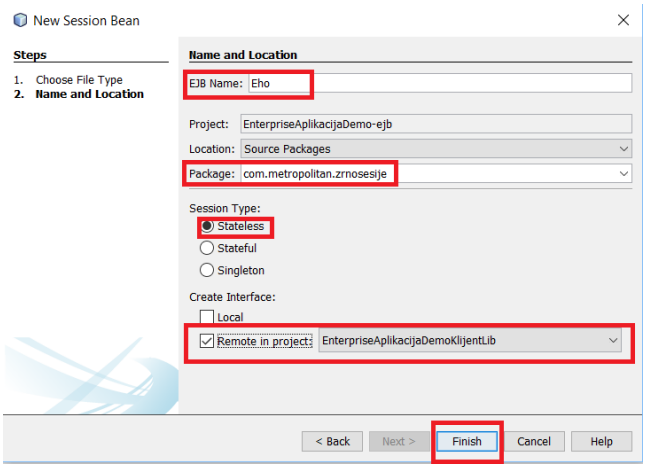
Prethodne verzije Java EE platforme zahtevale su lokalne interfejse kada su EJB zrna i njihovi klijenti izvršavani na istoj JVM. Od verzije Java EE 6, lokalni interfejsi su opcionalni.

Nije potrebno kreirati bilo kakav interfejs za zrna sesije ukoliko će im pristupati klijenti koji se izvršavaju na istoj JVM.

DODATNA RAZMATRANJA ZA ZRNA SESIJE

Akcenat je na distribuiranim sistemima i klijent će da se izvršava na udaljenoj JVM.

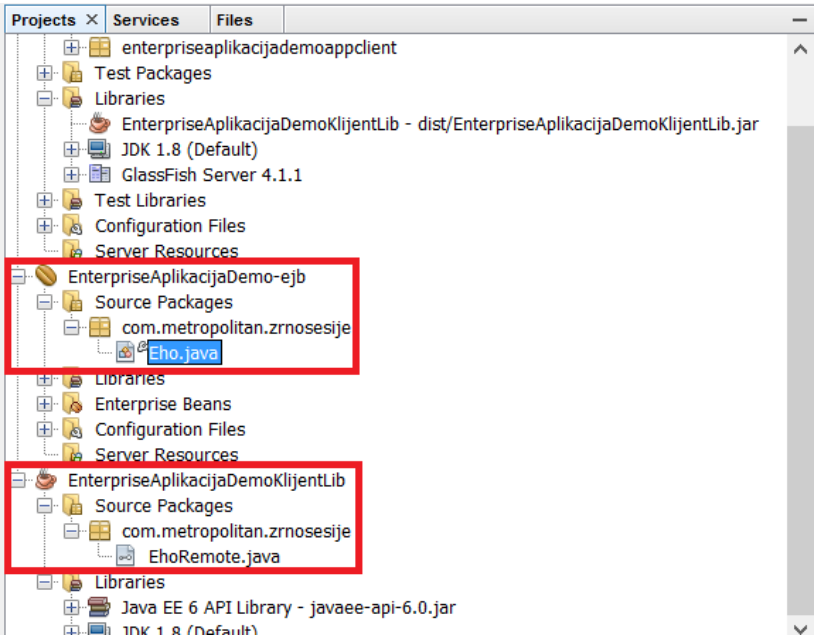
Za potrebe ovog primera biće kreirano zrno sesije koje ne mora da održava stanje konverzacije njegovih klijenata i biće definisano kao stateless zrno. Budući da je akcenat na distribuiranim sistemima, klijent će da se izvršava na udaljenoj Java virtuelnoj mašini. U tom kontekstu neophodno je kreirati udaljeni interfejs umesto lokalnog. Izborom udaljenog interfejsa, razvojno okruženje će zahtevati specificiranje klijentove biblioteke u okviru koje će udaljeni interfejs biti dodat. Klijentova biblioteka je dodata kao zavisnost istovremeno u EJB i klijent projekte. Upravo je ovo razlog zašto je bilo neophodno kreirati *Java biblioteku klasa* ranije. Po osnovnim podešavanjima, klijent *Java biblioteka klasa* biće ponuđena kada se izabere remote interfejs u prozoru prikazanim na sledećoj slici.



Slika 1.15 Podešavanja zrna sesije koje se kreira [izvor: autor]

Na prethodnoj slici je moguće primetiti da je u prozoru *New Session Bean* bilo neophodno specificirati i naziv *zrna sesije,* kao i naziv paketa kojem će zrno da pripada.

Klikom na dugme Finish, završava se kreiranje datoteke zrna sesije. Zrno je kreirano u projektu EJB modula, a udaljeni interfejs je kreiran u projektu klijentove biblioteke (sledeća slika).



Slika 1.16 Kreirano zrno sesije i udaljeni interfejs [izvor: autor]

KOD ZRNA SESIJE

Kreiranjem zrna sesije automatski je generisan inicijalni kod.

Nakon kreiranja datoteke *zrna sesije*, na način prikazan u prethodnom izlaganju, došlo je do automatskog kreiranja inicijalnog koda *zrna sesije* koji se čuva u ovoj datoteci i koji je priložen sledećim listingom:

```
package com.metropolitan.zrnosesije;

import javax.ejb.Stateless;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class Eho implements EhoRemote {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

Iz koda je moguće primetiti da je kreirana prazna klasa koja je već obeležena anotacijom ***@Stateless,*** koja ukazuje na zrno bez stanja, i koja implementira udaljeni interfejs. Kreirani interfejs je takođe prazan i obeležen je anotacijom ***@Remote***, koja ukazuje da se radi o udaljenom interfejsu. Inicijalni kod ovog interfejsa je priložen sledećim listingom.

```
package com.metropolitan.zrnosesije;
```

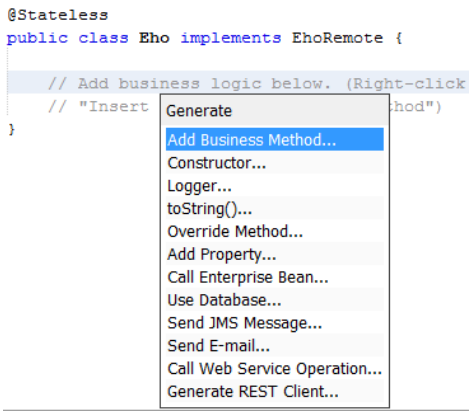
```
import javax.ejb.Remote;

/**
 *
 * @author Vladimir Milicevic
 */
@Remote
public interface EhoRemote {

}
```

Postojanje udaljenog ili lokalnog interfejsa je opravdano iz razloga što klijent zrna sesije nikada ne poziva metodu zrna direktno. Umesto toga oni dobijaju referencu klase koja implementira udaljeni ili lokalni interfejs i pozivaju metodu u ovoj klasi.

DODAVANJE POSLOVNE LOGIKE

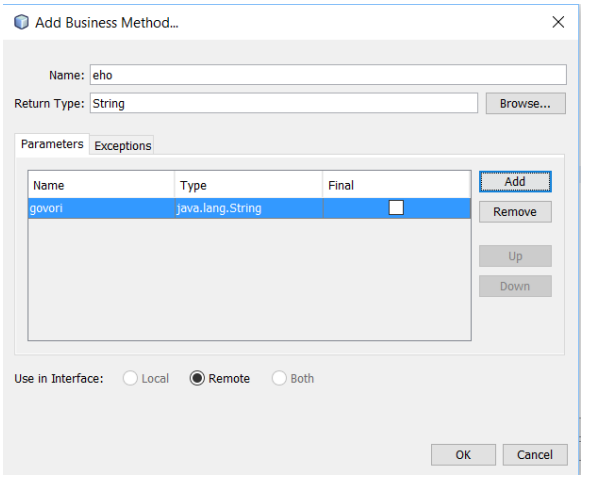
Sada je inicijalni kod moguće konkretizovati odgovarajućom programskom logikom.

Implementacija udaljenog (ili lokalnog) interfejsa sekreira automatski EJB kontejnerom kada se angažuje konkretno zrno. Implementacija vrši izvesna procesiranja pre poziva metode zrna sesije. Pošto metoda mora da bude definisana i u zrnu i u interfejsu, neophodno je dodavanje potpisa metode na oba mesta. Međutim, savremena razvojna okruženja, pa tako i NetBeans IDE, u velikoj meri olakšavaju i skraćuju ovaj posao. Jednostavnim desnim klikom na izvorni kod zrna i izborom opcije *Insert Code | Add Business Method* (sledeća slika) omogućeno je istovremeno kreiranje metode i u zrnu i u odgovarajućem interfejsu.



Slika 1.17 Kreiranje poslovne metode za zrno sesije [izvor: autor]

U nastavku, definiše se naziv metode, povratni tip i lista parametara. Ovaj proces je veoma pojednostavljen primenom čarobnjaka koji je prikazan sledećom slikom. Nakon toga metoda je prisutna i u zrnu i u interfejsu što je moguće primetiti u listinzima ispod slike 18.



Slika 1.18 Kreiranje poslovne metode [izvor: autor]

```
@Stateless
public class Eho implements EhoRemote {

    @Override
    public String eho(String govori) {
        return null;
    }
    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

```
@Remote
public interface EhoRemote {

    String eho(String govori);

}
```

VIDEO MATERIJALI

Ovaj deo lekcije biće zaokružen odgovarajućim video materijalom.

EJB - Introduction

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EJB Stateless Session Bean Example

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Pristup klijenta zrnu

EJB KLIJENT

Nakon osnovnih podešavanja i kreiranja zrna moguće je fokusirati se na EJB klikente.

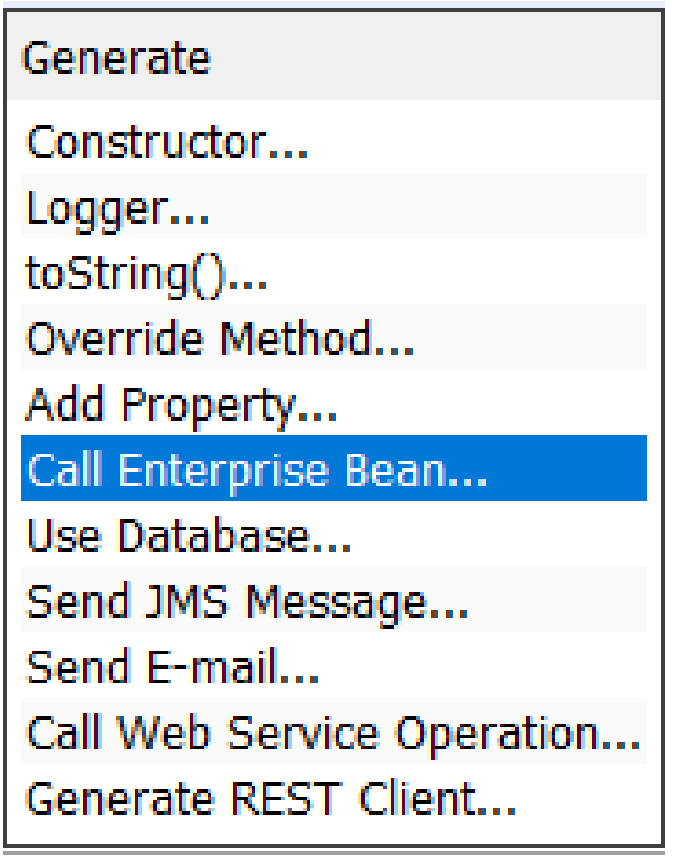
Nakon osnovnih podešavanja i kreiranja zrna moguće je fokusirati se na *EJB* klijente. Za trenutak, neophodno je vratiti se na metodu eho() zrna sesije koja po osnovnim podešavanjima vraća vrednost ***null.*** Za potrebe primera biće neophodno dati konkretan zadatak ovoj metodi i neka ona kao rezultat vraća konkretan ***String*** kao u kodu koji je priložen u narednom izlaganju i kojim se modifikuje početna metoda.

```
@Override
    public String eho(String govori) {
        return "Cuje se eho:" + govori;
    }
```

Sada ima potpunog smisla uposliti klijente EJB zrna da pristupaju odgovarajućem zrnu.

Distribuirani klijenti moraju da koriste projekat *Java biblioteke klasa* koji sadrži udaljeni interfejs koji je upravo iz ovog razloga prethodno i kreiran i dodat u klijentov projekat. Klijentova biblioteka sada sadrži udaljeni interfejs za kreirano zrno sesije i sve je spremno za pozivanje EJB metode.

Kod klijenta mora da sadrži referencu na instancu klase koja implementira udaljeni interfejs zrna. Primenom savremenih razvojnih okruženja ovaj zadatak je pojednostavljen, naročito ako se koristi NetBeans IDE. Desnim klikom na kod klase *Main,* projekta klijenta , u editoru razvojnog okruženja, bira se opcija *Call Enterprise Bean*. Navedeno je prikazano sledećom slikom.

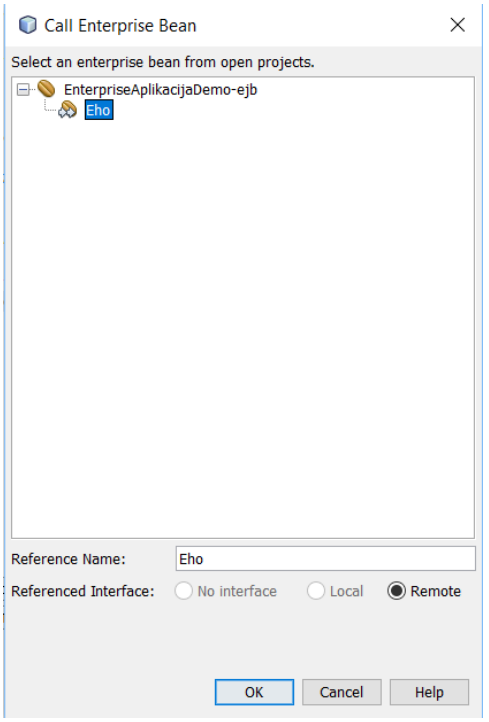


Slika 2.1 Umetanje koda za potivanje EJB zrna [izvor: autor]

IZBOR ZRNA

U nastavku je neophodno izabrati zrno kojem će klijent da se obraća.

U nastavku je neophodno izabrati zrno kojem će klijent da se obraća. Klikom na opciju *Call Enterprise Bean,* koja je prikazana na prethodnoj slici, otvara se prozor u kojem je neophodno izabrati odgovarajuće EJB zrno kojem će se obraćati kreirani klijent. Navedeno je prikazano sledećom slikom:



Slika 2.2 Izbor EJB zrna [izvor: autor]

Klikom na dugme OK u kod klase *Main*, klijenta zrna, dodata je varijabla *eho* tipa *EhoRemote* koja je obeležena anotacijom ***@EJB.*** Ovom anotacijom se vrši umetanje instance udaljenog interfejsa tokom vremena izvršavanja.

Na samom kraju je neophodno pozvati metodu eho() za udaljeni interfejs u okviru glavne metode klase *Main* projekta klijenta. Klijent će biti kompletiran i imaće izgled koji odgovara kodu iz sledećeg listinga.

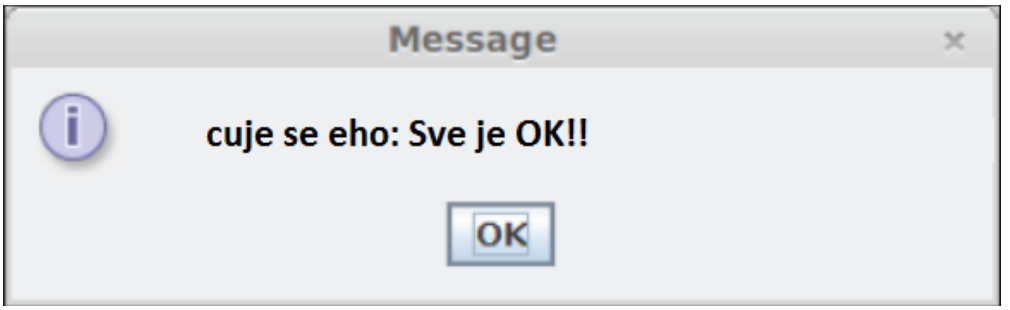
```
package enterpriseaplikacijademoappclient;

import com.metropolitan.zrnosesije.EhoRemote;
import javax.ejb.EJB;
import javax.swing.JOptionPane;

/**
 *
 * @author Vladimir Milicevic
 */
public class Main {

    @EJB
    private static EhoRemote eho;

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null,
                eho.eho("Sve je OK!!!"));
```

```
    }

}
```

IZVRŠAVANJE DISTRIBUIRANOG KLIJENTA

Nakon završetka definicije klijenta neophodno je testirati urađeni posao.

Sada je moguće testirati kreirani primer. Desnim klikom na projekat Enterprise aplikacije i izborom opcije *Run* pokreće se kreirana aplikacija i rezultat se može videti na sledećoj slici.



Slika 2.3 Poziv metode eho() klijentom EJB zrna [izvor: autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Klijenti angažovani na ovaj način predstavljaju svu prednost primene *Java Web Start* tehnologije. *Java Web Start* aplikacije izvršavaju se na klijentovim stanicama. Međutim mogu biti izvršene i preko *URL* - a. *Web Start URL* za enterprise *NetBeans* aplikaciju sastoji se, po osnovnim podešavanjima, od naziva glavnog projekta, praćenog nazivom projekta EJB klijenta. U konkretnom slučaju, za ovaj projekat URL bi glasio: http://localhost:8080/EnterpriseAplikacijaDemo/EnterpriseAplikacijaDemo-app-client.

Poglavlje 3

Zrna sesije - upravljanje transakcijama

KONTROLA TRANSAKCIJA

Moguće je koristiti dodatna podešavanja za bolju kontrolu transakcija.

Iako je pomenuto, u prethodnom izlaganju, da je jedna od najvećih prednosti primene EJB automatsko vođenje računa o transakcijama, moguće je obaviti dopunska podešavanja sa ciljem kvalitetnije kontrole nad procesom upravljanja transakcijama.

Transakcije omogućavaju izvođenje svih koraka u metodi i / ili, u slučaju pada nekog koraka (na primer izbacivanja izuzetka), vraćanje na stanje pre promena koje su učinjene u metodi.

Primarno, ovde će biti reči o podešavanju ponašanja zrna u slučaju poziva neke od njegovih metoda tokom izvršavanja transakcije. Da li će metoda postati deo aktuelne transakcije ili će aktuelna transakcija biti otkazana i biti kreirana nova za metode zrna, zavisi od podešavanja koja su određena primenom anotacije @TransactionAttribute.

Anotacija ***@TransactionAttribute*** omogućava programerima da kontrolišu kako se ponašaju metode *EJB* zrna koje se pozivaju kada je transakcija u toku i, takođe, koje se pozivaju kada je transakcija nije u toku. Ova anotacija poseduje jedinstven atribut ***value*** koji ukazuje na način kako će se ponašati metoda zrna u oba navedena slučaja.

Sledećom tabelom pokazane su različite vrednosti, koje mogu biti pridružene anotaciji ***@TransactionAttribute***

@TransactionAttribute - vrednost	Poziv metode kada je transakcija u toku.	Poziv meode kada transakcija nije u toku
`TransactionAttributeType.MANDATORY`	Metoda postaje deo aktuelne transakcije	`TransactionRequiredException` je izbačen
`TransactionAttributeType.NEVER`	`RemoteException` je izbačen	Metoda se izvršava bez ikakve podrške transakcije
`TransactionAttributeType.NOT_SUPPORTED`	Klijentova transakcija je privremeno suspendovana, metoda se izvršava bez podrške transakcije, zatim se klijentova transakcija nastavlja	Metoda se izvršava bez ikakve podrške transakcije
`TransactionAttributeType.REQUIRED`	Metoda postaje deo aktuelne transakcije	Nova transakcija je kreirana za metodu
`TransactionAttributeType.REQUIRES_NEW`	Klijentova transakcija je privremeno suspendovana, metoda se izvršava bez podrške transakcije, zatim se klijentova transakcija nastavlja	Nova transakcija je kreirana za metodu
`TransactionAttributeType.SUPPORTS`	Metoda postaje deo aktuelne transakcije	Metoda se izvršava bez ikakve podrške transakcije

Slika 3.1 Vrednosti anotacije @TransactionAttribute sa opisom ponašanja [izvor: autor]

PRIMENA ANOTACIJE @TRANSACTIONATTRIBUTE

Anotacija može biti primenjena na EJB zrno ili njegove metode.

Anotacija ***@TransactionAttribute*** može biti upotrebljena na dva načina:

- Anotacijom je moguće obeležiti klasu EJB zrna;
- Anotacijom je moguće obeležiti metoduEJB zrna.

Ukoliko se anotacija koristi za obeležavanje klase *EJB zrna* deklarisano transakciono ponašanje će biti primenjeno na sve metode u klasi. U suprotnom, ukoliko se anotacija koristi za obeležavanje pojedinačne metode, navedeno ponašanje će biti karakteristično samo za obeleženu metodu.

Ukoliko je u klasi EJB zrna anotacija @TransactionAttribute primenjena i na klasu i na metode, prednost će imati anotacija koja je deklarisana na niviou metode.

Ukoliko nije specificiran atribut transakcije za metodu obeleženu anotacijom @TransactionAttribute, podrazumevana vrednost će bitiTransactionAttributeType.REQUIRED.

Sledećim listingom je prikazana modifikacija klase EJB zrna, aktuelnog primera, u kojem je implementirana ***@TransactionAttribute*** anotacija.

```
package com.metropolitan.zrnosesije;

import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class Eho implements EhoRemote {

    @Override
    @TransactionAttribute(
     TransactionAttributeType.REQUIRES_NEW)
    public String eho(String govori) {
        return "cuje se eho:" + govori;
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

Iz priloženog je moguće primetiti da je metoda jednostavno podešena obeležavanjem pomoću anotacije ***@TransactionAttribute*** sa odgovarajućom vrednošću za ***TransactionAttributeType.*** Ukoliko je cilj da ovo ponašanje bude implementirano na sve metode klase, dovoljno je ovu anotaciju pomeriti na nivo klase *EJB zrna*.

VIDEO - JAVA EE TRANSACTION ATTRIBUTES

Doskusija o EJB upravljanju transakcijama će biti zaokružena priloženim video materijalom

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 4

Implementacija AOP sa presretačima

ASPEKTNO - ORIJENTISANO PROGRAMIRANJE

EJB 3.0 predstavlja mogućnost implementacije AOP -a putem presretača (interceptors).

Ponekad je neophodno izvršiti neku programsku logiku po scenarijima: pre, posle ili za vreme izvršavanja glavne logike metode. Na primer, šalje se log poruka svaki put kada se ulazi u metodu ili se ona napušta sa ciljem lakšeg praćena grešaka i izuzetaka.

Veoma elegantan način za realizovanje rešenja za ovakav tip problema jeste dodavanje još malo koda na početak i kraj svake metode, implementirajući logiku za slanje log poruka. Ovakav pristup sa sobom nosi i izvesne nedostatke, a oni se ogledaju u višestrukoj implementaciji logike. To ima za posledicu da ukoliko se želi da izvrši izmena ili uklanjanje neke funkcionalnosti, to je neophodno učiniti na više mesta u programu.

Aspektno - orijentisano programiranje (AOP) je paradigma programiranja koja uspešno rešava navedene probleme kroz mehanizme implementacije logike, po scenarijima: pre, posle ili za vreme izvršavanja glavne logike metode, u izdvojenoj klasi.

EJB 3.0 predstavlja mogućnost implementacije AOP -a putem presretača (interceptors).

Implementacija AOP - a putem presretača se odvija u dva koraka:

- kodiranje klase presretala;
- obeležavanje *EJB* za presretanje anotacijom @Interceptors.

O navedenom će posebno biti reči u narednom izlaganju.

KLASA PRESRETAČ

Presretač je standardna Java klasa koja poseduje metodu obeleženu anotacijom AroundInvoke.

Presretač je standardna Java klasa koja poseduje jednu metodu sa sledećim potpisom:

```
@AroundInvoke
public Object methodName(InvocationContext invocationContext) throws
Exception
```

Ono što je moguće primetiti jeste da je metoda obeležena anotacijom ***@AroundInvoke*** koja ističe metodu kao metodu presretača. Parametar ***InvocationContext*** može biti iskorišćen da pribavi informaciju od presretnute metode kao što su: naziv, parametri, klasa u kojoj je deklarisana i tako dalje. ***InvocationContext*** takođe poseduje metodu ***proceed()*** koja ukazuje kada je potrebno izvršiti glavnu logiku.

Sledećom tabelom je priložena lista najčešće korišćenih ***InvocationContext*** metoda.

Za kompletnu listu metoda neophodno je konsultovati Java dokumentaciju - Help | JavaDoc References | Java (TM) EE 7
Specification APIs.

Naziv metode	Opis
`getMethod()`	Metoda vraća instancu klase java.lang.reflect.Method koja može biti upotrebljena za proveru presretnute metode.
`getParameters()`	Vraća niz objekata koji sadrže parametre prosleđene presretnutoj metodi.
`getTarget()`	Vraća objekat koji sadrži pozvanu metodu. Povratna vrednost je java.lang.Object.
`proceed()`	Poziva presretnutu metodu.

Slika 4.1 Lista najčešće korišćenih InvocationContext metoda. [izvor: autor]

PRIMER JEDNOSTAVNE KLASE PRESRETAČA

Sledi kod koji ilustruje jednostavnog presretača.

U nastavku, za lakše razumevanje izgleda i funkcionisanja klase presretača, biće neophodno priložiti primer koda jedne takve klase.

```
import java.lang.reflect.Method;
import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

/**
 *
 * @author Vladimir Milicevic
 */

public class LoggingInterceptor {

    @AroundInvoke
    public Object logMethodCall(
            InvocationContext invocationContext)
            throws Exception {
        Object interceptedObject
                = invocationContext.getTarget();
        Method interceptedMethod
                = invocationContext.getMethod();
```

```
        System.out.println("Ulazak "
                + interceptedObject.getClass().getName() + "."
                + interceptedMethod.getName() + "()");
        Object o = invocationContext.proceed();
        System.out.println("Izlazak "
                + interceptedObject.getClass().getName() + "."
                + interceptedMethod.getName() + "()");
        return o;
    }
}
```

U prethodnom primeru se šalje log poruka aplikativnom serveru neposredno pre i neposredno posle izvršavanja glavne logike. Svrhe ovakvog postupanja je ste olakšavanje debagovanja aplikacije.

Zbog jednostavnosti ovde je korišćeno System.out.
println za izlazne poruke. U praksi bi verovatnije bilo upotrebljeni Java Logging API ili Log4j.

Prvo što se radi u metodi presretača jeste prihvatanje reference na objekat i metodu koji se presreću. Tada se obezbeđuje izlazna poruka (log) kojom se ukazuje na objekat i metodu koji su presretnuti. Ovaj kod se izvršava pre nego što se dozvoli izvršavanje presretnute metode kroz poziv *invocationContext.proceed().* Povratna vrednost metode se čuva u posebnoj varijabli, a zatim se izvršava još malo logike nakon izvršavanja presretnute metode. Konačno, metoda vraća povratnu vrednost poziva *invocationContext.proceed().*

OBELEŽAVANJE EJB ZRNA @INTERCEPTORS ANOTACIJOM

Da bi EJB metoda mogla da bude presretnuta mora da bude obeležena anotacijom @Interceptors.

Da bi *EJB* metoda mogla da bude presretnuta neophodno je da bude obeležena anotacijom ***@Interceptors.*** Ova anotacija poseduje jedinstveni atribut koji ukazuje na niz klasa presretača. Ovaj atribut sadrži sve presretače koje se izvršavaju pre i / ili posle poziva metode.

Anotacija ***@Interceptors*** može biti upotrebljena na nivou metode i u tom slučaju je primenjiva isključivo na jednu metodu koju je obeležila. Takođe, anotacijom može biti obeležena i cela klasa. U tom slučaju, anotacija je primenjiva na sve metode klase *EJB* zrna.

Za potrebe analize i demonstracije ponovo će akcenta biti na konkretnom primeru. Već poznato zrno sesije *Eho.java* biće modifikovano na način da će njegova *eho()* metoda biti presretana klasom *LoggingInterceptor.java* koja je priložena u prethodnom izlaganju.

Slika 4.2 Praćenje presretanja u monitoru servera GlassFish [izvor: autor]

```
package com.metropolitan.zrnosesije;

import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.interceptor.Interceptors;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class Eho implements EhoRemote {

// Add business logic below. (Right-click in editor and choose
// "Insert Code > Add Business Method")

@Interceptors({LoggingInterceptor.class})
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)

    public String eho(String govori) {
        return "Cuje se: " + govori;
    }

}
```

Poglavlje 5

EJB Timer servis

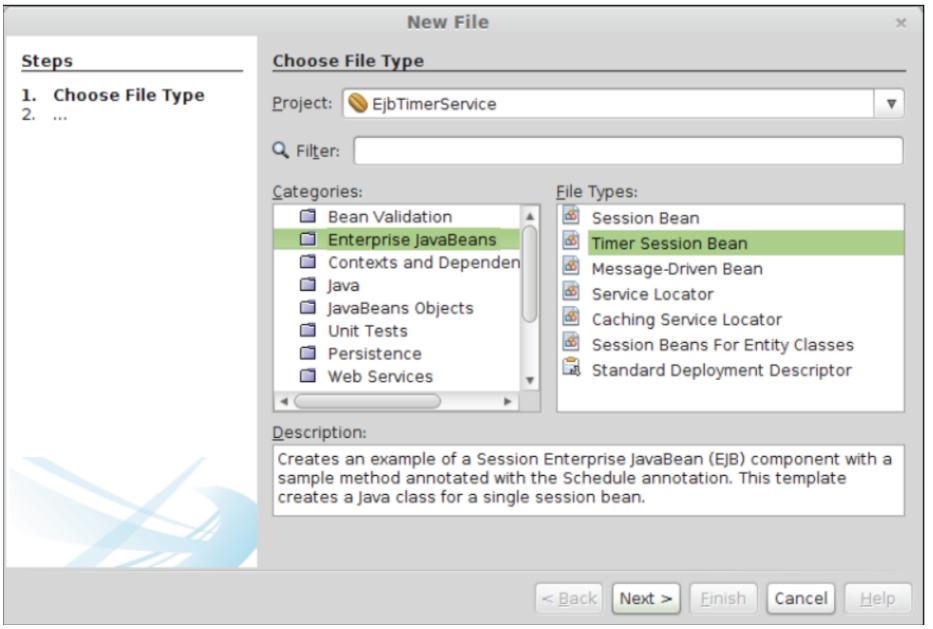
TIMER SESSION BEAN

Omogućavanje periodičnog izvršavanja neke logike bez eksplicitnog pozivanja bilo koje metode.

Zrna bez stanja (stateless), kao i zrna vođena porukama (message driven), o kojima će kasnije biti više govora, mogu da poseduju metodu koja se izvršava automatski u regularnim intervalima. Funkcionalnost je veoma značajna ukoliko se želi obezbeđivanje periodičnog izvršavanja neke logike (dnevno, nedeljno, svaki drugi sat i tako dalje) bez eksplicitnog pozivanja bilo koje metode. Ova funkcionalnost je omogućena primenom EJB Timer servisa.

Da bi ovaj servis mogao da bude korišćen, neophodno je upotrebiti anotaciju @Schedule za specificiranje kada će metoda biti pozvana. Primenom savremenog razvojnog okruženja, kakav je i NetBeans IDE, moguće je obezbediti pogodne alate za pomoć u realizaciji vremenskih servisa.

Sledećom slikom je prikazan čarobnjak, kojeg obezbeđuje NetBeans IDE, razvojno okruženje, za pomoć pri kreiranju i implementaciji vremenskih servisa. Na slici se lako da primetiti kako se koristi kategorija *Enterprise JavaBeans* za izbor tipa datoteke koji se kreira, a u ovom konkretnom slučaju je to Timer Session Bean.



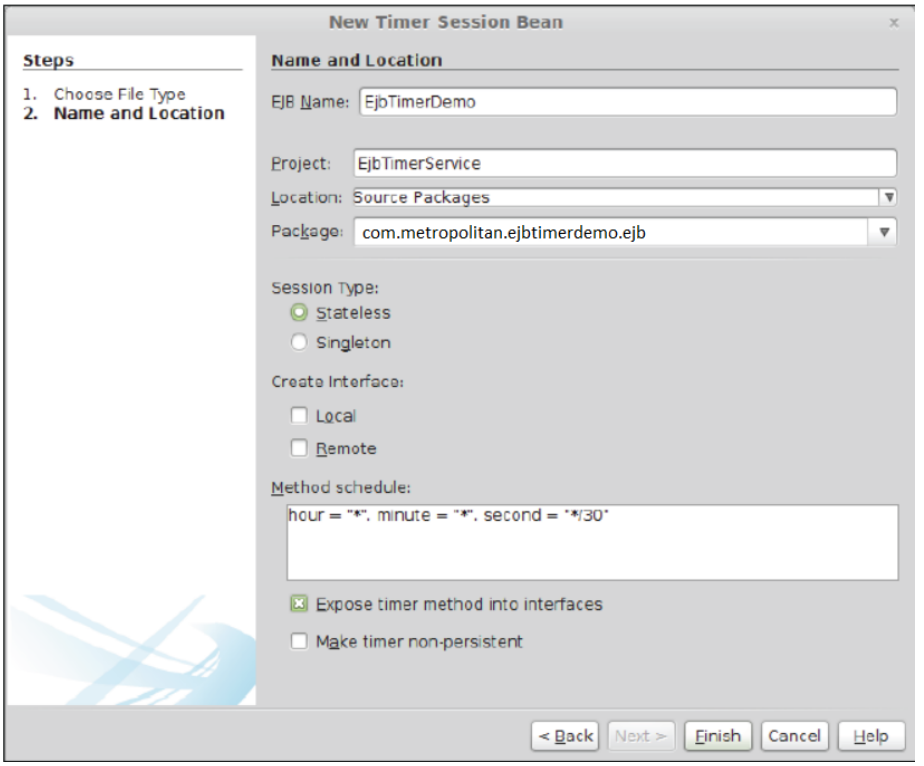
Slika 5.1 Kreiranje Timer Session Bean datoteke [izvor: autor]

U narednom izlaganju biće objašnjena dodatna podešavanja ovog zrna u prozoru koji se pojavljuje nakom klika na dugme Next sa Slike1.

ANOTACIJA @SCHEDULE

Neophodno je dodatno podesiti zrno sesije koje se kreira.

Klikom na dugme Next, pogledati prethodnu sliku, otvara se nov prozor u kojem je neophodno dodatno podesiti zrno sesije koje se kreira. Navedeni prozor je prikazan sledećom slikom.

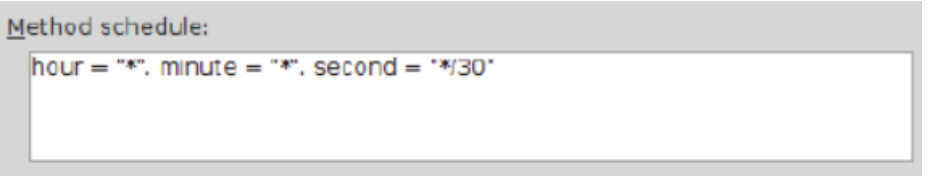


Slika 5.2 Dodatno podešavanje za Timer Session Bean [izvor: autor]

U ovom delu je neophodno dodeliti odgovarajući naziv zrnu, a u konkretnom slučaju naziv glasi *EjbTimerDemo.* U nastavku moguće je primetiti da prozor sadrži informacije poput: naziva projekta, u okviru kojeg se kreira datoteka *EjbTimerDemo.java,* lokacije i naziva paketa kojem ova Java klasa pripada.

Takođe, biće izabrana opcija *Stateless* koja ukazuje da se radi o zrnu bez stanja. Za potrebe ovog primera nije potrebno čekirati tip interfejsa, mada kada se budu kreirale *distribuirane aplikacije*, biraće se *Remote* opcija.

U polju za unos teksta *Method schedule* unose se atributi i vrednosti koji će biti pridruženi anotaciji ***@Schedule*** kada se posmatrano zrno generiše (sledeća slika).



Slika 5.3 Parametri i vrednosti buduće anotacije @Schedule

U nastavku je neophodno izvršiti klik na dugme Finish nakon čega će zrno biti generisano. O detaljima zrna će biti više reči u narednom izlaganju.

PRIMER KODA ZRNA I TESTIRANJE

U nastavku se analizira generisani kod i rezultati njegovog izvršavanja.

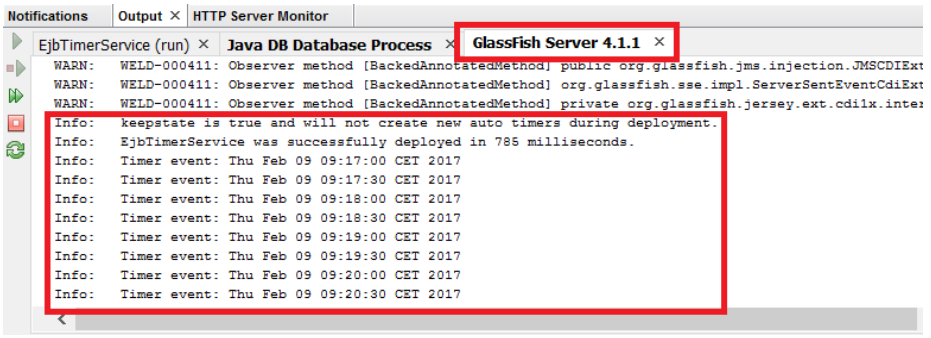
Kao što je istaknuto u prethodnom izlaganju, sa završetkom podešavanja, vođenog prikazanim *NetBeans IDE* čarobnjakom, dolazi do generisanja koda zrna vremenskog servisa. Sledećim listingom je prikazan kod ovog zrna.

```
package com.metropilitan.ejbtimer.ejb;

import java.util.Date;
import javax.ejb.Schedule;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
@LocalBean
public class EjbTimerDemo {

    @Schedule(hour = "*", minute = "*", second = "*/30")
    public void myTimer() {
        System.out.println("Timer event: " + new Date());
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

Kada se obrati posebna pažnja na priloženi listing, moguće je primetiti da je uatomatski implementirana metoda ***myTimer()*** koja je obeležena anotacijom ***@Schedule.*** Anotacija obuhvata tri atributa sa odgovarajućim vrednostima:

- *hour* = "*" znači da će metoda biti pozivana svakog sata;
- *minute*= "*" znači da će metoda biti pozivana svakog minuta;
- *second* = "*/30" znači da će metoda biti pozivana na svakih 30 sekundi;

Sada je moguće uposliti kreirani primer i pokrenuti ga u okviru *NetBeans IDE* razvojnog okruženja. Odlaskom na tab koji pripada aplikativnom serveru *GlassFish* moguće je uočiti kako se kreirani vremenski servis izvršava, periodično: na 30s, jedan minut i jedan sat (sledeća slika).



Slika 5.4 Izvršavanje vremenskog servisa [izvor: autor]

VIDEO MATERIJAL ZA EJB TIMER

Izlaganje u ovom delu lekcije će biti zaokruženo odgovarajućim video materijalom.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 6

Kreiranje zrna sesija iz JPA entiteta

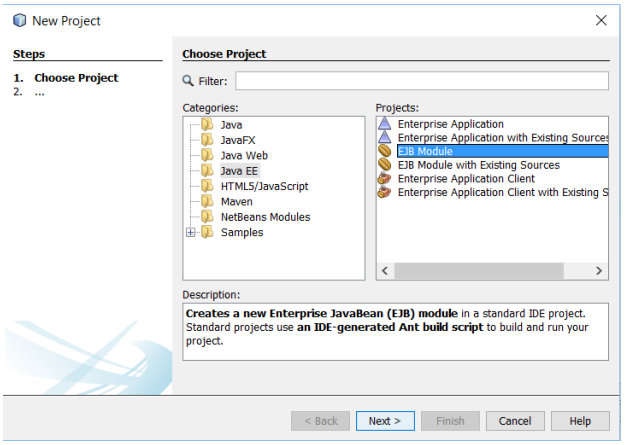
GENERISANJE ZRNA BEZ STANJA

NetBeans razvojno okruženje omogućava generisanje zrna bez stanja iz postojećih JPA entiteta.

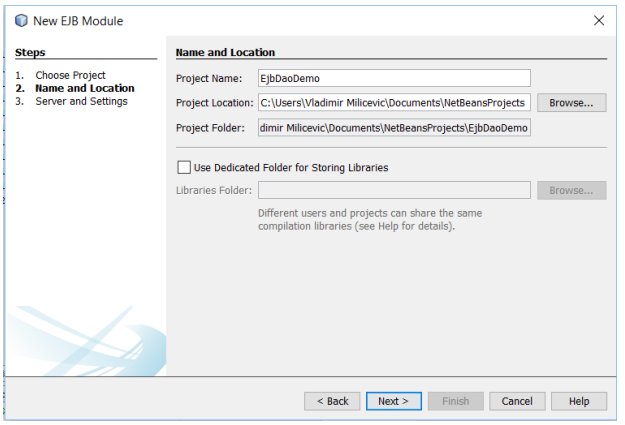
Veoma dobra i korisna osobina koju poseduju savremena Java razvojna okruženja jeste automatizacija brojnih programerskih zadataka. Na ovaj način programeri su fokusirani direktno na rešavanje konkretne programske logike bez upuštanja u brojne dugotrajne i zamarajuće zadatke.

NetBeans razvojno okruženje omogućava generisanje zrna bez stanja (Stateless) iz postojećih JPA entiteta. Na ovaj način, kreirana zrna se ponašaju kao dobro poznati DAO (Data AccessObjects) objekti za pristupanje podacima. Posebno je značajno što na ovaj način je moguće iskoristiti postojeće šeme baza podataka za kompletno generisanje nivoa za pristup podacima, u okviru distribuiranih Java EE aplikacija, bez pisanja jedne jedine linije programskog koda.

Za potpuno razumevanje i demonstraciju navedene problematike biće uključen konkretan primer. Primer podrazumeva kreiranje EJB projekta na način što će u razvojnom okruženju biti izabrana opcija *File | New Project,* a zatim iz kategorije *Java EE* je neophodno izabrati tip projekta *EJB Module.* Navedeno je prikazano Slikom 1. Nakon toga je neophodno dodeliti odgovarajući naziv projektu, a to je urađeno na način prikazan Slikom 2. Iz prozora , prikazanog Slikom 2 se odlazi na prozor u kojem je dovoljno kliknuti na dugme Finish za potvrđivanje podešavanja demonstriranih priloženim slikama.



Slika 6.1 Kreiranje novog EJB Module projekta

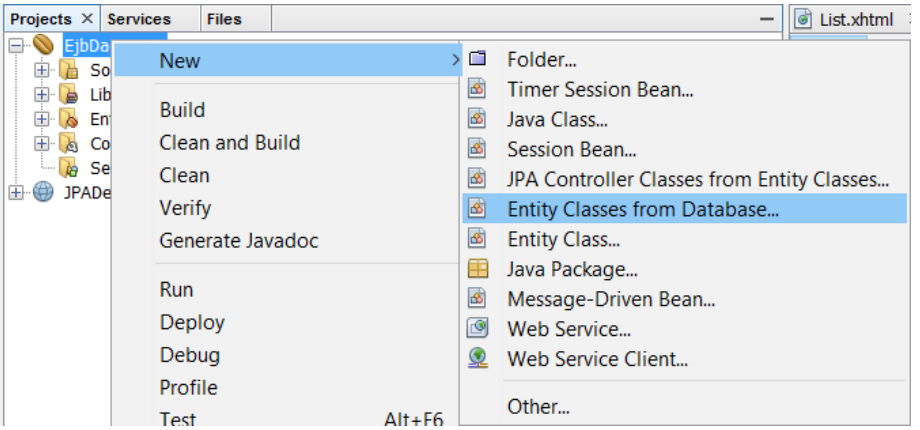


Slika 6.2 Dodela naziva novom EJB Module projektu [izvor: autor]

IZBOR JPA ENTITETA

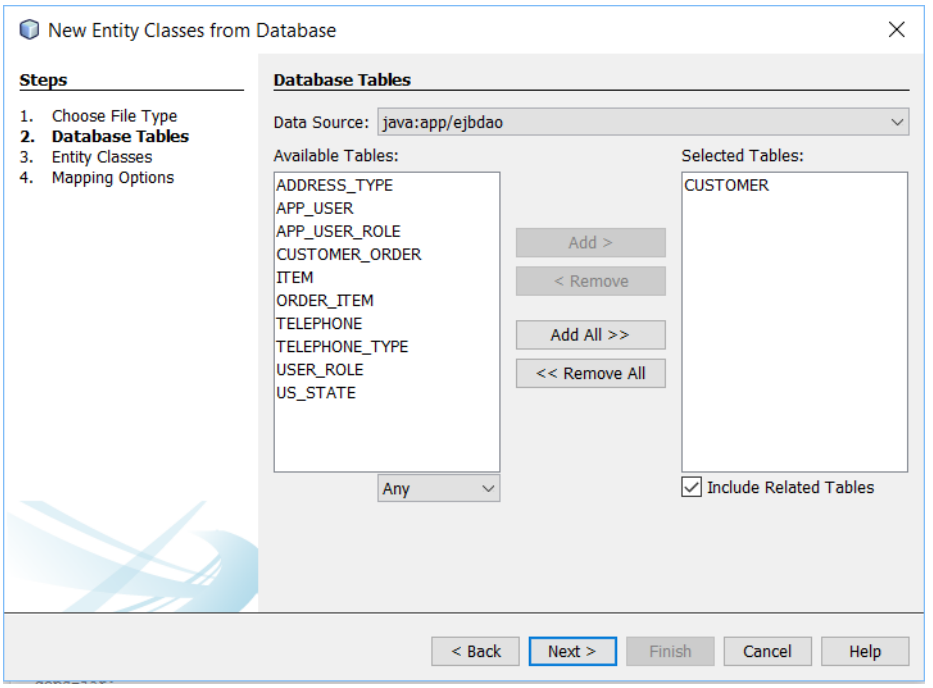
U nastavku je neophodno izabrati JPA entitete iz kojih će biti generisana zrna sesija.

Nakon obavljenih zadataka iz prethodne sekcije, kreiran je EJB Module projekat. koji je neophodno povezati sa postojećom bazom podataka za upotrebu JPA entiteta koji su već kreirani nad njom. JPA entiteti mogu biti ručno programirani, a mogu biti i generisani iz tabela paze podataka kao što je to urađeno u prethodnoj lekciji. Taj pristup će biti upotrebljen i u ovom slučaju. Izborom opcije *New*, pa zatim *Entity Classes from Database* biće kreirano i dodato u projekat onoliko JPA entiteta koliko je potrebno kreirati zrna (sledeća slika).



Slika 6.3 Dodavanje JPA Entiteta u kreirani EJB Module projekat [izvor: autor]

Izborom odgovarajućeg izvora podataka, odnosno baze podataka, dobija se lista tabela nad kojima je moguće kreirati *JPA entitete* (sledeća slika). Zbog jednostavnosti, u ovom slučaju će biti izabrana samo jedna tabela, a ceo primer će detaljno biti urađena na časovim vežbi.



Slika 6.4 Izbor tabele za dodavanje JPA entiteta u EJB Module projekat [izvor: autor]

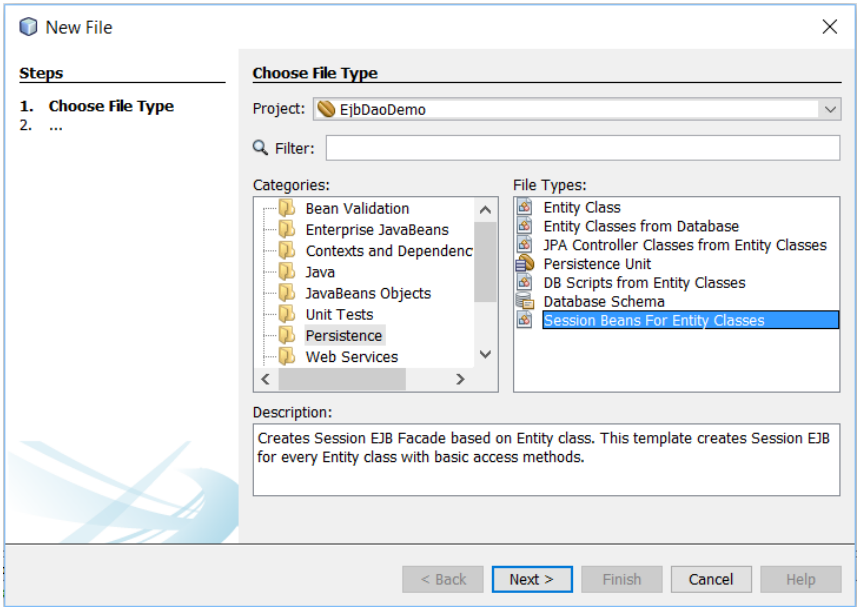
Kao što je moguće primetiti sa slike, izabrana je jedna tabela *CUSTOMER* i klikom na *Next* se inicira proces kojim se završava kreiranje JPA entiteta i njegovo dodavanje u kreirani *EJB Module* projekat.

U sledećem izlaganju je neophodno pokazati kako se iz ovog JPA entiteta automatski generiše zrno sesije.

KREIRANJE ZRNA SESIJE

Iz postojećeg JPA entiteta se kreira zrno sesije.

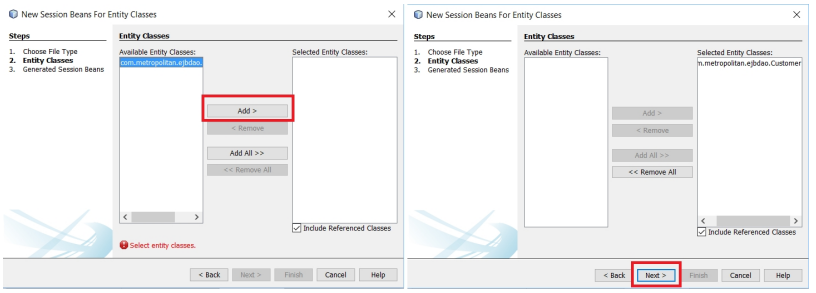
Prethodni korak je podrazumevao obezbeđivanje *JPA entiteta* neophodnih za generisanje zrna sesije. U nastavku, u kreiranom projektu, pristupa se generisanju zrna sesije na osnovu prisutnog JPA entiteta *Customer.java.* U razvojnom okruženju se bira opcija *File | New File,* a zatim se iz kategorije *Persistence* bira opcija *Session Beans For EntityClasses.* Navedeno je prikazano sledećom slikom.



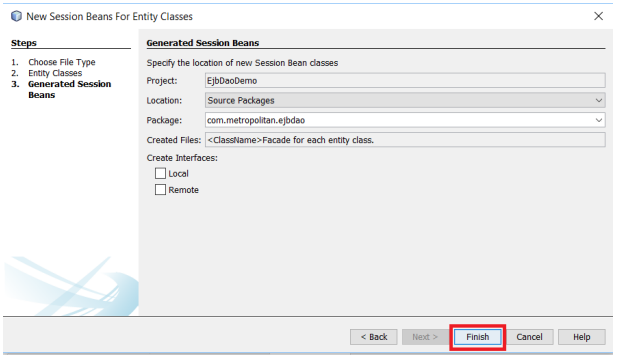
Slika 6.5 Izbor kreiranja zrna sesije iz JPA entiteta [izvor: autor]

Klikom na dugme Next, otvara se prozor u kojem se nalazi lista dostupnih *JPA entiteta* iz kojih je moguće kreirati zrna sesije.

Sledeća slika pokazuje prozor pre (levo) i posle (desno) izbora JPA entiteta od kojeg se generiše EJB zrno sesije. Proces teče u sledećim koracima: bira se *JPA entitet* iz liste dostupnih JPA entiteta, klikom na dugme *Add* on se selektuje za generisanje zrna, klikom na *Next* započinje kreiranje zrna iz izabranih *JPA entiteta*. Na kraju (slika 7) definiše se paket zrna i bira *Finish*.



Slika 6.6 Izbor JPA entiteta za kreiranje zrna sesije [izvor: autor]

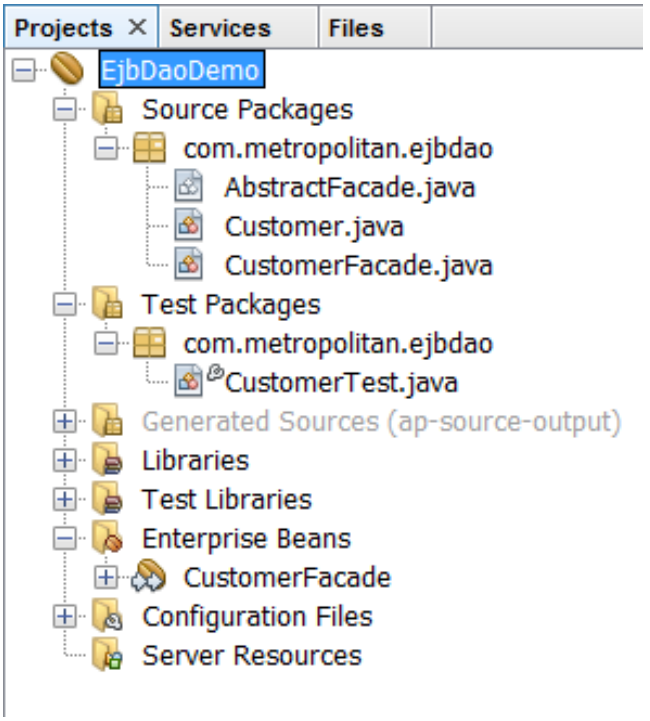


Slika 6.7 Izbor paketa za kreirano zrno sesije [izvor: autor]

KREIRANE DATOTEKE PROJEKTA

U nastavku je neophodno izvršiti analizu generisanog koda.

U nastavku izlaganja je neophodno prikazati i analizirati kod koji je generisan kao posledica koraka koji su izvođeni praćenjem instrukcija čarobnjaka razvojnog okruženja *NetBeans IDE*. Sledećom slikom je prikazana struktura projekta sa svim pripadajućim datotekama, koje su od interesa za dalju analizu i izlaganje.



Slika 6.8 Struktura kreiranog EjbDaoDemo projekta [izvor: autor]

Dalja analiza će se posebno fokusirati na sledeće datoteke:

- *Customer.java;*
- *AbstractFacade.java;*
- *CustomerFacade.java*.

Prva datoteka, o kojoj će najmanje biti govora jer je ta tema već obrađena, odgovara JPA entitetu *Customer* koji je kreiran nad tabelom *CUSTOMER* aktuelne baze podataka. Zbog kompletne slike, prilaže se listing ove datoteke.

```
package com.metropolitan.ejbdao;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author Vladimir Milicevic
 */
@Entity
@Table(name = "CUSTOMER")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c")
    , @NamedQuery(name = "Customer.findByCustomerId", query = "SELECT c FROM
Customer c WHERE c.customerId = :customerId")
    , @NamedQuery(name = "Customer.findByFirstName", query = "SELECT c FROM
Customer c WHERE c.firstName = :firstName")
    , @NamedQuery(name = "Customer.findByMiddleName", query = "SELECT c FROM
Customer c WHERE c.middleName = :middleName")
    , @NamedQuery(name = "Customer.findByLastName", query = "SELECT c FROM Customer
c WHERE c.lastName = :lastName")
    , @NamedQuery(name = "Customer.findByEmail", query = "SELECT c FROM Customer c
WHERE c.email = :email")})
public class Customer implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "CUSTOMER_ID")
    private Integer customerId;
    @Size(max = 20)
    @Column(name = "FIRST_NAME")
    private String firstName;
    @Size(max = 20)
    @Column(name = "MIDDLE_NAME")
    private String middleName;
    @Size(max = 20)
    @Column(name = "LAST_NAME")
    private String lastName;
    //
@Pattern(regexp="[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?
:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?",
message="Invalid email")//if the field contains email address consider using this
annotation to enforce field validation
    @Size(max = 30)
    @Column(name = "EMAIL")
    private String email;

    public Customer() {
```

```
    }

    public Customer(Integer customerId) {
        this.customerId = customerId;
    }

    public Integer getCustomerId() {
        return customerId;
    }

    public void setCustomerId(Integer customerId) {
        this.customerId = customerId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getMiddleName() {
        return middleName;
    }

    public void setMiddleName(String middleName) {
        this.middleName = middleName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (customerId != null ? customerId.hashCode() : 0);
        return hash;
    }
```

```
    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not
set
        if (!(object instanceof Customer)) {
            return false;
        }
        Customer other = (Customer) object;
        if ((this.customerId == null && other.customerId != null) ||
(this.customerId != null && !this.customerId.equals(other.customerId))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "com.metropolitan.ejbdao.Customer[ customerId=" + customerId + " ]";
    }

}
```

DATOTEKA ABSTRACTFACADE.JAVA

Posebna pažnja se obraća na datoteku koja apstraktnoj klasi.

U ovom trenutku je najvažnije napomenuti da sve klase koje odgovaraju generisanim zrnima sesija nasleđuju apstraktnu klasu *AbstractFacade.java* koja je takođe generisana poštovanjem i izvođenjem koraka u *Session Beans for Entity Classes* čarobnjaku razvojnog okruženja *NetBeans IDE*. Ova klasa je zaslužna za obezbeđivanje metoda za izvođenje CRUD (Create,Read, Update, Delete) operacija nad entitetima. Sledećim listingom je priložen kod ove apstraktne klase.

```
package com.metropolitan.ejbdao;

import java.util.List;
import javax.persistence.EntityManager;

/**
 *
 * @author Vladimir Milicevic
 */
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }
```

```
    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }

    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

Kada se pogleda sam kod moguće je primetiti da ova klasa predstavlja mnogo više nego što je "fasada" za EntityManager. Obmotavanjem svih poziva unutar zrna sesije dobija se velika prednost njegove primene koja se ogleda uupravljanju transakcijama i distribuiranom kodu.

Generisana metoda *create()* je zadužena za kreiranje novih entiteta, dok su metode *edit()* i *remove()* zadužene za ažuriranje i uklanjanje entiteta, respektivno. Metoda *find()* pronalazi entitet sa zadatim primarnim ključem, a metoda *findAll()* vraća listu svi entiteta preuzetih

iz baze podataka. Metodom *findRange()* moguće je vratiti određeni podskup entiteta iz baze podataka koji određen nizom celih brojeva (parametar metode). Metoda *count()* prebrojava i vraća broj entiteta iz baze podataka.

DATOTEKA ZRNA SESIJE

Klasa generisanog zrna sesije nasleđuje klasu AbstractFacade.

Kao što je istaknuto u prethodnom izlaganju, sve generisane klase, koje odgovaraju zrnima sesije, nasleđuju apstraktnu klasu AbstractFacade. Takav je slučaj i sa klasom *CustomerFacade.java* koja je generisana na opisani način iz *JPA entiteta Customer.java.* Sledećim listingom je priložen kod kojim je implementirana logike generisanog zrna sesije.

```
package com.metropolitan.ejbdao;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class CustomerFacade extends AbstractFacade<Customer> {

    @PersistenceContext(unitName = "EjbDaoDemoPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public CustomerFacade() {
        super(Customer.class);
    }

}
```

Iz priloženog koda je moguće primetiti da je generisano zrno sesije veoma jednostavno. Zrno uključuje instancu tipa *EntityManager,* a to ima za prednost korišćenje umetnutih resursa u njegovu inicijalizaciju. Zrno, takođe, uključuje metodu *getEntityManager()* zamišljenu da bude pozivana roditeljskom klasom i ima pristup *EntityManager instanci* zrna sesije. Na kraju, konstruktor zrna sesije poziva konstruktor roditeljske klase koji, pomoću generika, inicijalizuje *entityClass* instancu za roditeljsku klasu.

U zrno sesije je moguće dodati i druge metode.

Nedostatak ovog pristupa predstavlja činjenica da ukoliko je neophodno ponovo generisati zrno sesije, sve postojeće korisničke metode će biti izgubljene. Da bi ovo bilo prevaziđeno, moguće je naslediti zrna sesije i dodatne metode čuvati u klasama potomcima.

Poglavlje 7

EJB - Pokazni primer (45 min)

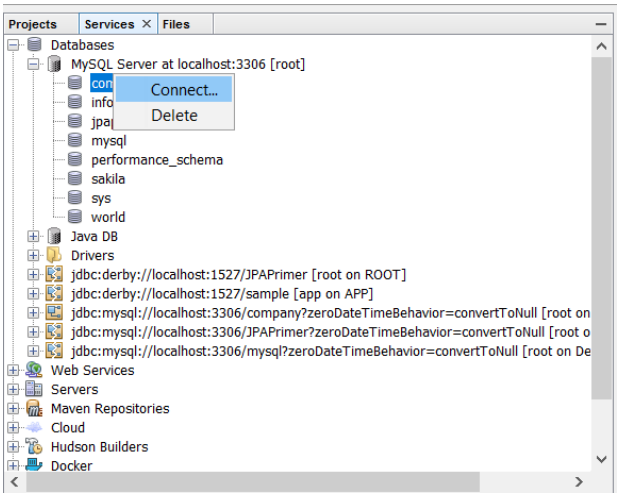
POSTAVKA, KREIRANJE PROJEKTA I POVEZIVANJE NA DB (45 MIN)

Cilj časova vežbi je vežbanje na računaru kreiranja EJB zrna sesije iz JPA entiteta.

Zadatak:

1. Kreirati EJB Module projekat PP8;
2. Iskoristiti bazu podataka *company* iz prethodnih vežbi i nad njom kreirati JPA entitete u projektu PP8;
3. Iz kreiranih JPA entiteta kreirati EJB zrna sesije.

Kao što je moguće videti iz zahteva zadatka, na poznati način je prvo neophodno kreirati nov projekat, koji će se zvati PP8. Zatim je neophodno kliknuti na tab *Services* i izvršiti konektovanje na bazu podataka *company* (sledeća slika).



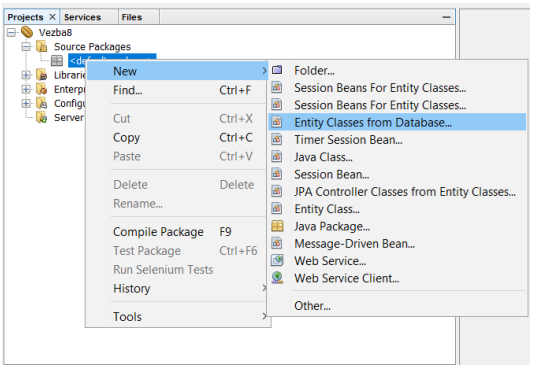
Slika 7.1 Povezivanje na bazu podataka [izvor: autor]

Ukoliko baza ne postoji, neophodno je izvršiti njeno kreiranje i dodati joj odgovarajuće tabele. Fajl sa SQL upitima za generisanje tabela je priložen kao dodatni materijal na kraju vežbi i nalazi se u SQL podfolderu projekta.

KREIRANJE JPA ENTITETA

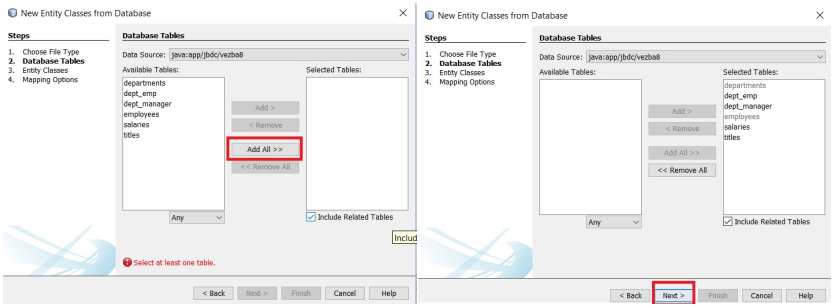
U sledećem koraku, kreiraju se JPA entiteti

Zatim je neophodno, na takođe poznati način kreirati JPA entitete. Kao na slici, bira se opcija *New,* a zatim *Entity Classes From Database*.

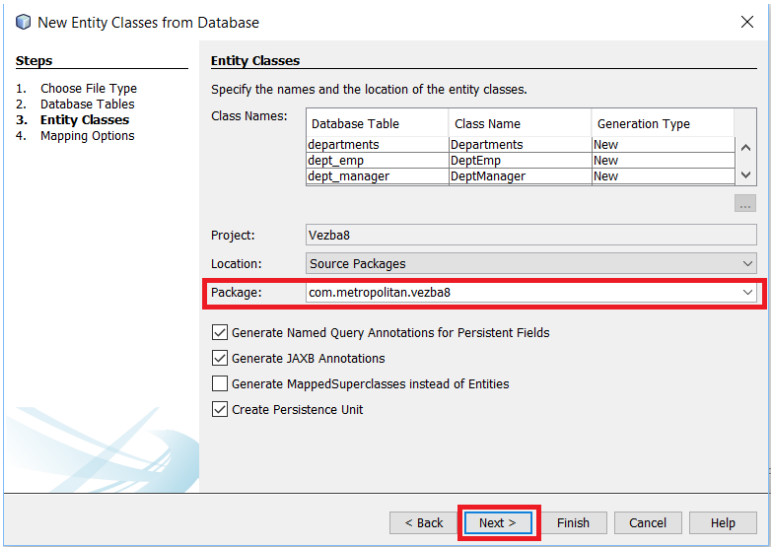


Slika 7.2 Kreiranje JPA entiteta iz tabela baze podataka [izvor: autor]

Nakon toga, otvara se prozor u kojem biramo dostupne tabele baze podataka *company* od kojih je neophodno kreirati JPA entitete. Navedeno je pokazano Slikom 3. Klikom na *Next* otvara se novi prozor za izbor paketa.



Slika 7.3 Izbror tabela baze podataka za JPA entitete [izvor: autor]

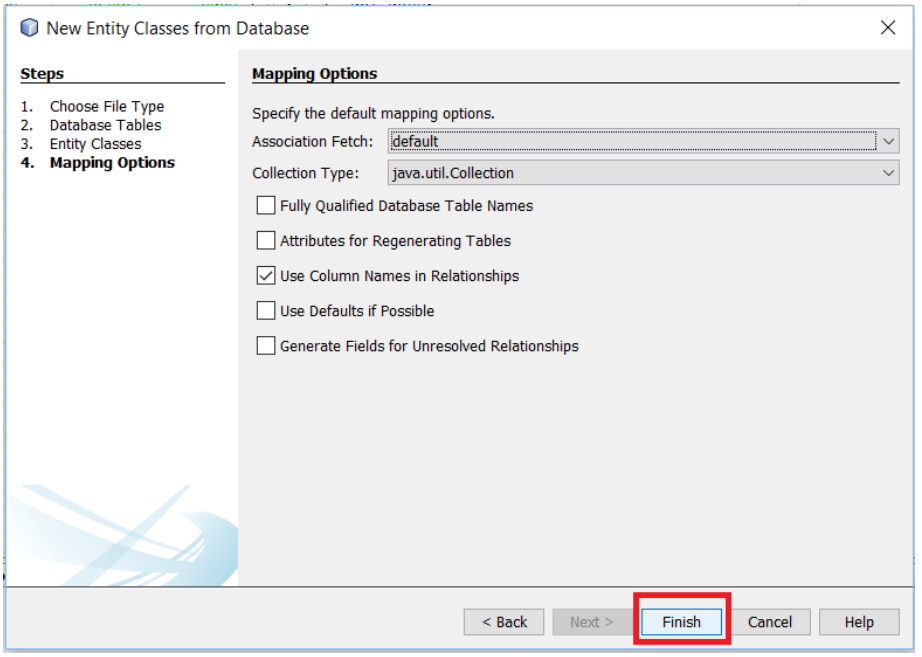


Slika 7.4 Izbor paketa za JPA entitete [izvor: autor]

DOVRŠAVANJE DEFINICIJE JPA ENTITETA

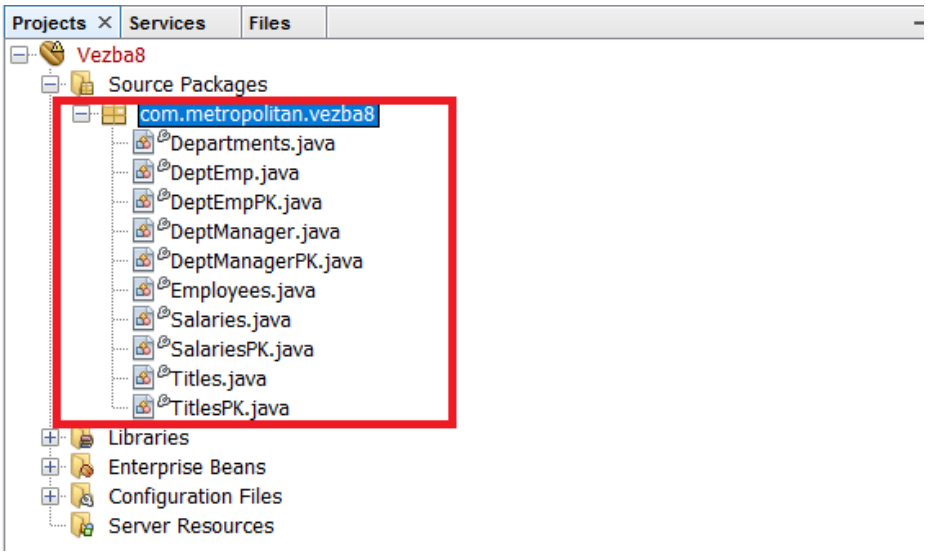
Izvode se poslednji koraci u kreiranju JPA entiteta.

Klikom na *Next*, u prozoru prikazanim prethodnom slikom, otvara se sledeći prozor u kojem se klikom na *Finish* kompletira proces kreiranja JPA entiteta iz tabela baze podataka *company*.



Slika 7.5 Kraj čarobnjaka za kreiranje JPA entiteta [izvor: autor]

Sledećom slikom je prikazana trenutna hijerarhija projekta sa kreiranim klasama JPA entiteta.

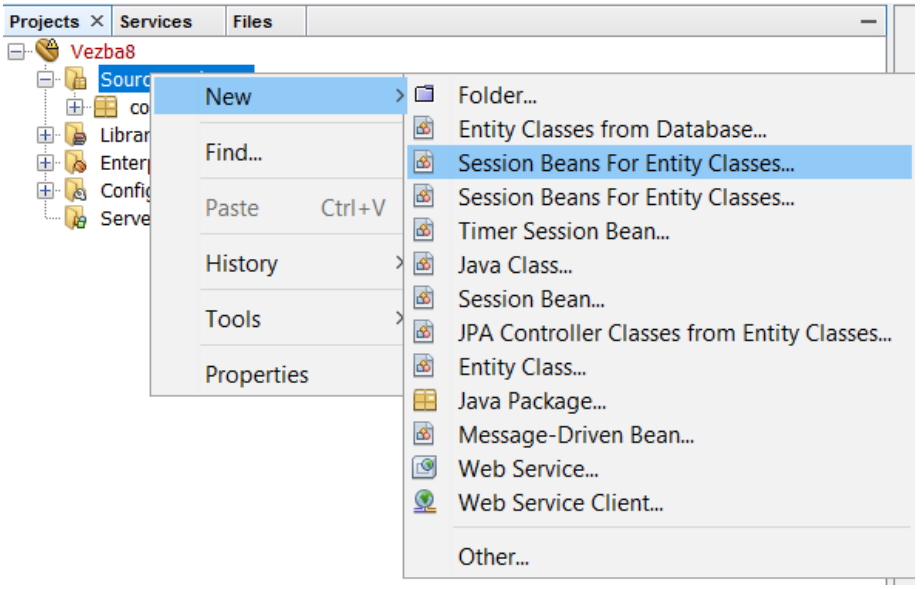


Slika 7.6 Klase JPA entiteta u projektu Vezba8 [izvor: autor]

KREIRANJE EJB ZRNA SESIJA IZ JPA ENTITETA

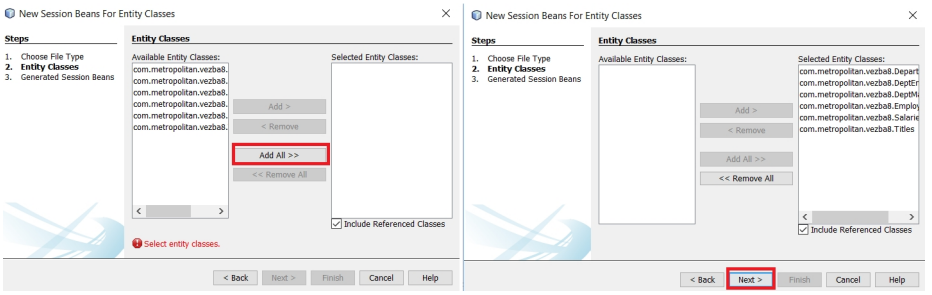
Sledeći zadatak je kreiranje EJB zrna sesija iz JPA entiteta.

Iz postojećih klasa entiteta neophodno je kreirati zrna sesija primenom razvojnog okruženja NetBeans IDE. Neophodno je postupiti kao na sledećoj slici. U postojećem projektu *Vezba8,* desnim klikom na *Source Packages,* bira se opcija *New,* a zatim i *Session Beans For Entity Classes.*



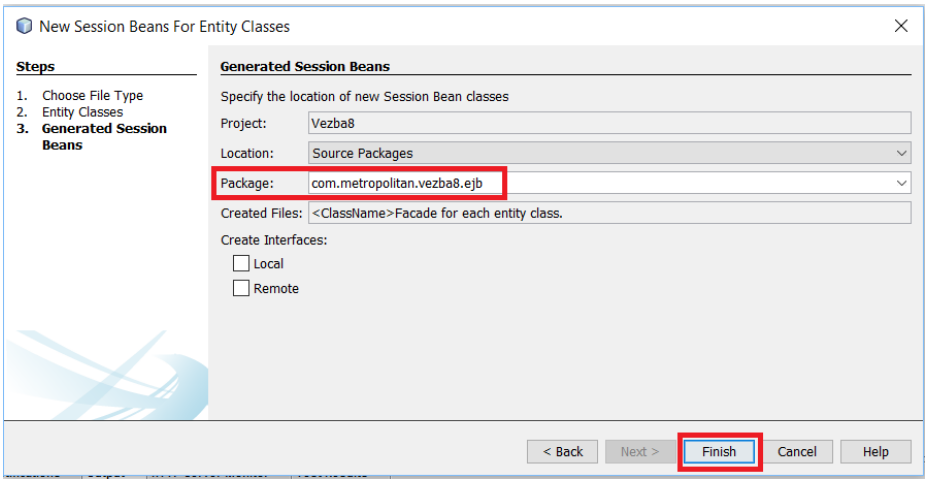
Slika 7.7 Kreiranje EJB zrna sesija iz JPA entiteta - početak [izvor: autor]

Nakon toga se otvara prozor u kojem se bira iz liste dostupnih JPA entiteta od kojih će biti kreirana zrna sesije.



Slika 7.8 Izbor JPA entiteta za kreiranje zrna sesije [izvor: autor]

Klikom na *Next,* otvara se prozor za izbor paketa i kraj zadataka koji se izvode ovim čarobnjakom.

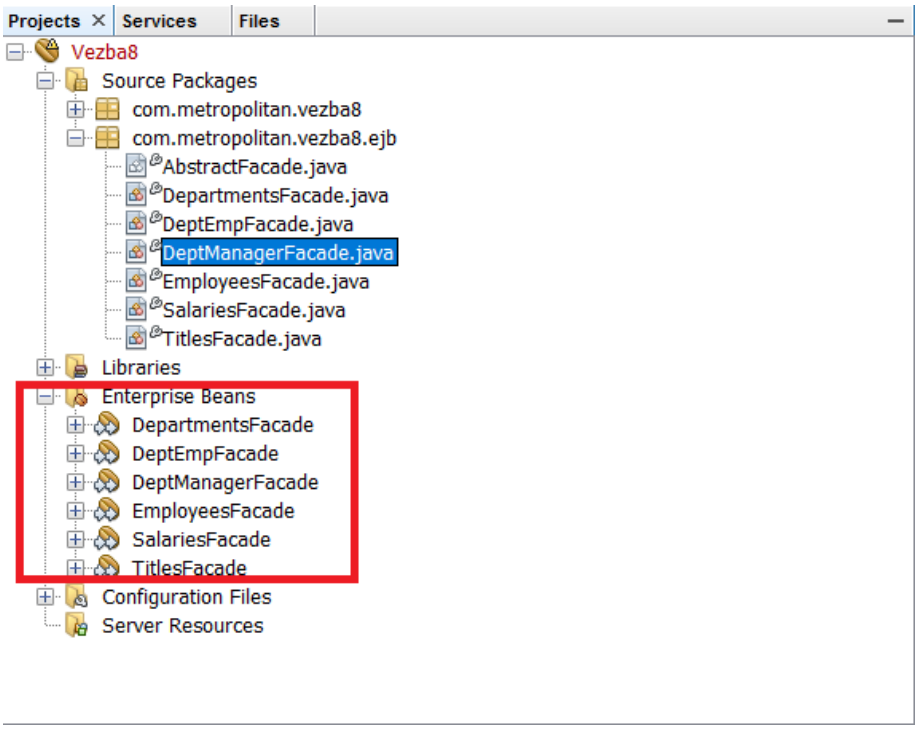


Slika 7.9 Izbor JPA entiteta za kreiranje zrna sesije - kraj [izvor: autor]

LISTING KLASE ABSTRACTFACADE

Slede listinzi klasa kreiranih EJB zrna

Sledećom slikom su prikazane klase EJB zrna koje se nalaze u projektu *Vezbe8.* Sledećom slikom je prikazana potpuna hijerarhija projekta.



Slika 7.10 Klase EJB zrna [izvor: autor]

U prvom koraku je priložena apstraktna klasa *AbstractFacade* o kojoj je dosta bilo reči u materijalima predavanja i koju nasleđuju sve ostale klase zrna sesije.

```
package com.metropolitan.pp8.ejb;

import java.util.List;
import javax.persistence.EntityManager;

/**
 *
 * @author Vladimir Milicevic
 */
public abstract class AbstractFacade<T> {

    private Class<T> entityClass;

    public AbstractFacade(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    protected abstract EntityManager getEntityManager();

    public void create(T entity) {
        getEntityManager().persist(entity);
    }

    public void edit(T entity) {
        getEntityManager().merge(entity);
    }
```

```
    public void remove(T entity) {
        getEntityManager().remove(getEntityManager().merge(entity));
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }

    public List<T> findRange(int[] range) {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        q.setMaxResults(range[1] - range[0] + 1);
        q.setFirstResult(range[0]);
        return q.getResultList();
    }

    public int count() {
        javax.persistence.criteria.CriteriaQuery cq =
getEntityManager().getCriteriaBuilder().createQuery();
        javax.persistence.criteria.Root<T> rt = cq.from(entityClass);
        cq.select(getEntityManager().getCriteriaBuilder().count(rt));
        javax.persistence.Query q = getEntityManager().createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    }

}
```

LISTINZI KLASA EMPLOYEESFACADE I DEPARTMENTSFACADE

Sledi listing klasa EmployeesFacade i DepartmentsFacade.

Sledećim listingom je priložena klasa zrna sesije *EmployeesFacade.*

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.Employees;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
```

```
/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class EmployeesFacade extends AbstractFacade<Employees> {

    @PersistenceContext(unitName = "Vezba8PU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public EmployeesFacade() {
        super(Employees.class);
    }

}
```

Sledećim listingom je priložena klasa zrna sesije *DepartmentsFacade*.

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.Departments;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class DepartmentsFacade extends AbstractFacade<Departments> {

    @PersistenceContext(unitName = "Vezba8PU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public DepartmentsFacade() {
        super(Departments.class);
    }

}
```

LISTINZI KLASA TITLESFACADE I SALARIESFACADE

Sledi listing klasa TitlesFacade i SalariesFacade.

Sledećim listingom je priložena klasa zrna sesije *TitlesFacade*.

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.Titles;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class TitlesFacade extends AbstractFacade<Titles> {

    @PersistenceContext(unitName = "Vezba8PU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public TitlesFacade() {
        super(Titles.class);
    }

}
```

Sledećim listingom je priložena klasa zrna sesije *SalariesFacade*.

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.Salaries;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class SalariesFacade extends AbstractFacade<Salaries> {

    @PersistenceContext(unitName = "Vezba8PU")
```

```
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public SalariesFacade() {
        super(Salaries.class);
    }

}
```

LISTINZI KLASA DEPTMANAGERFACADE I DEPTEMPFACADE

Slede listinzi klasa DeptManagerFacade i DeptEmpFacade.

Sledećim listingom je priložena klasa zrna sesije *DeptManagerFacade*.

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.DeptManager;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class DeptManagerFacade extends AbstractFacade<DeptManager> {

    @PersistenceContext(unitName = "Vezba8PU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public DeptManagerFacade() {
        super(DeptManager.class);
    }

}
```

Sledećim listingom je priložena klasa zrna sesije *DeptEmpFacade.*

```
package com.metropolitan.pp8.ejb;

import com.metropolitan.vezba8.DeptEmp;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

/**
 *
 * @author Vladimir Milicevic
 */
@Stateless
public class DeptEmpFacade extends AbstractFacade<DeptEmp> {

    @PersistenceContext(unitName = "Vezba8PU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public DeptEmpFacade() {
        super(DeptEmp.class);
    }

}
```

Poglavlje 8

Individualna vežba 8

INDIVIDUALNA VEŽBA (135 MIN)

Pokušajte sami

Zadatak 1 (65 min):

1. Kreirajte bazu podataka *fakultet;*
2. Baza podataka ima sledeće tabele: *Student* (sifra_st, ime, adresa) i *Status* (sifra_statusa, naziv_statusa,sifra_st);
3. Naziv statusa je: tradicionalni ili Internet;
4. Kreirati veb aplikaciju sa JPA entitetima nad ovom bazom podataka;
5. Kreirati EJB Module projekat sa zrnima sesije nad kreiranim JPA entitetima.

Zadatak 2 (70 min):

1. Postupite obrnuto - kreirajte entitete: Student i Status;
2. Kreirajte bazu podataka fakultet2;
3. Kreirajte tabele iz entiteta - direktnim JPA inženjeringom;
4. Kreirati EJB Module projekat sa zrnima sesije nad kreiranim JPA entitetima.

Poglavlje 9

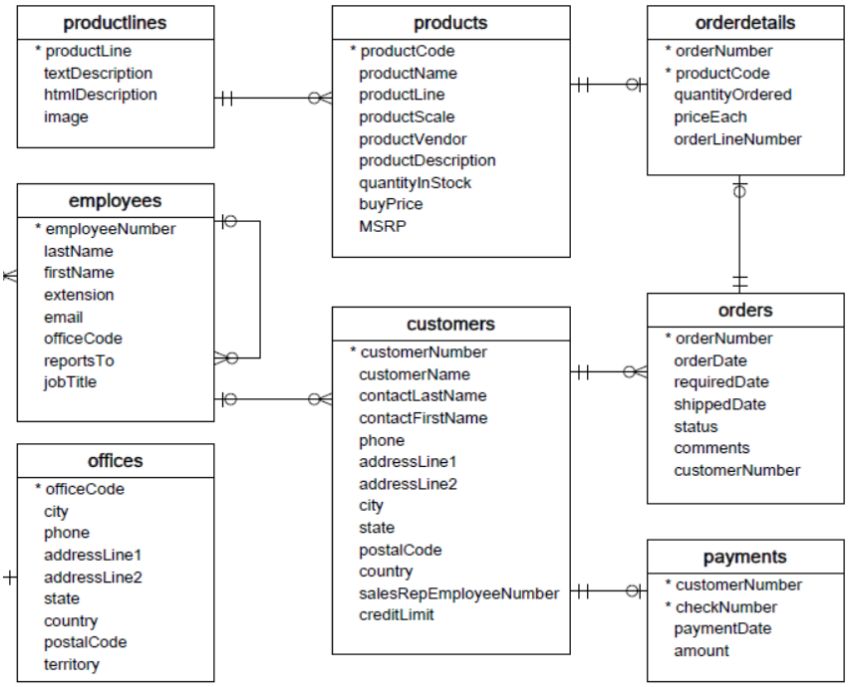
Domaći zadatak 8

ZADATAK 1 (120 MIN)

Izrada domaćeg zadatka - kreiranje EJB zrna sesije iz JPA entiteta.

Po uzoru na zadatak sa vežbi - kreirajte EJB zrna sesije iz JPA entiteta kreiranih nad priloženom šemom baze podataka:

- direktnim i
- obrnutim JPA inženjeringom..



Slika 9.1 Šema baze podataka za kreiranje EJB zrna sesije iz JPA entiteta. [izvor: autor]

Nakon obaveznog zadatka, studenti na email dobijaju različite zadatke od predmetnog asistenta.

Poglavlje 10

Zaključak

ZAKLJUČAK

Zaključna razmatranja lekcije

Na samom kraju, sledi kratak rezime tema obrađenih u ovoj lekciji.

Lekcija je na samom početku istakla da većina Java EE aplikacija mora da vodi računa o uobičajenim zahtevima kao što sutransakcije, bezbednost, skalabilnost i tako dalje. Upravo iz navedenog razloga je analiza bila bazirana na EJB (Enterpirise JavaBeans) koji dozvoljavaju programerima da se fokusiraju na razvoj poslovne logike bez potrebe za vođenjem računa o implementaciji navedenih zahteva.

Posebno je istaknuto da postoje dva tipa EJB zrna:

- zrna sesije (session beans);
- zrna vođena porukama (message-driven beans).

Zrna vođena porukama nisu bila od značaja za ovu lekciju pa je u akcenat bio na analizi i demonstraciji zrna sesije koja u velikoj meri pojednostavljuju razvoj programske logike na serverskoj strani. Vodeći se navedenim, akcenat u lekciji je bio na detaljnom proučavanju sledećih tema:

- Predstavljanje zrna sesije;
- Kreiranje zrna sesije;
- EJB upravljanje transakcijama;
- Primena aspektno - orijentisanog programiranja sa presretačima;
- EJB Timer servis;
- Generisanje zrna sesije iz JPA entiteta.

Savladavanjem ove lekcije student u potpunosti razume i vlada primenom zrna sesija u Java EE aplikacijama.

LITERATURA

Za pripremu lekcije korišćena je najnovija literatura.

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. HeffelFinger. 2015. Java EE7 Develpomnet With NetBeans 8, PACK Publishing

3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh JUneau. 2015. Java EE7 Recipes, Apress