TESTOVI – KI203

TEST 1

1. Koji od sledećih metoda su instans (objektni) metodi u java.lang.Thread? run, start, stop, suspend, resume, sleep, interrupt, zield, join?

- **run**, **sleep**, **yield**, **join**.

2. Ako petlja sadrži metod koji izbacuje InterruptedException, zašto treba da se petlja stavi u try-catch blok?

- Java zahteva da uhvatimo proverene izuzetke, jer ako je petlja koja sadrži takav metod van try-catch bloka, izvršenje niti se može nastaviti i u slučaju njenog prekidanja.

3. Kako postavljate prioritet za neku nit? Koji je početno određeni prioritet?

- Metodom **setPriority()** se određuje prioritet niti, a metodom **getPriority()** se dobija prioritet niti. JVM uvek uzima trenutnu nit u izvršenju kao nit sa najvišim prioritetom. Niži prioritet se koristi samo kada više nema niti sa višim prioritetom. Ako sve niti imaju isti prioritet, da bi se izbegla blokada sve niti tada dobijaju jednako CPU vreme.

4. Koje su koristi od primene pula niti?

- Kada se radi sa velikim brojem zadataka, preporučuje se upotreba "pula niti". On predstavlja idealan način za istovremeni rad sa puno zadataka.

5. Kako kreirate pul niti sa tri fiksne niti? Kako podnosite zadataka pulu niti? Kako znate da su svi zadaci završeni?

- Metod newFixedThreadPool(3) kreira pul sa 3 fiksne niti. Tačnije:

ExecutorService executor=new Executors.newFixedThreadPool(3);

Pomoću metode **execute()** se predaje zadatak pulu na izvršenje.
Metod **isTermminated()** vraća true ako su svi zadaci u pulu izvršeni.

6. Dajte nekoliko primera mogućih oštećenja resursa kada izvršavate program sa više niti. Kako sinhronizujete niti u konfliktu?

- Problem oštećenja podataka koji se javlja kada sve niti imaju simultani pristup istom izvoru pdataka može se prikazati kao:
  * nepredvidiv iznos krajnjeg rezultata, ukoilko su nam potrebne tačne i predvidive vrednosti (kamatni račun, balans stanja, ...i sl.) .
  * nekonzistentnost podataka, ... i sl.

Niti u konfliktu možemo da sinhronizujemo sinhonizacijom ključne reči (sinhronizacija upotrebom ključeva) ili sinhronizovanim iskazom (korišćenjem sinhronizovanog bloka).

7. Šta podrazumeva politika fer ponašanja?

- Ispravna (true) **politika fer ponašanja** podrazumeva (garantuje) da nit koja najduže čeka prva dobija ključ.

8. Kako kreirate uslov za neki ključ? Za šta služe metodi await(), siganal() i signalAll()?

- Uslovi su objekti koji su kreirani metodima **newCondition()** nad "Lock" objektima.
  Metod **await()** prouzrokuje da tekuća nit čeka dok se uslov ne ispuni.
  Metod **signal()** budi nit koja čeka.
  Metod **signalAll()** budi sve niti koje čekaju.

9. Šta bi se desilo ako bi while petlju u liniji 58 listinga ThreadCooperation zamenili sa if(balance < amount)?

- Upotreba if iskaza može da vodi do netačnog povlačenja novca.

10. Koje objekte biste koristili za animaciju?

- Za animacije se koriste tzv. "Timeline" objekti.

11. Mogu li da se metodi read() i write() u klasi Buffer simultano izvršavaju?

- Mogu. Zato što je bafer red sa strategijom FIFO (prvi ušao prvi izašao). Metod **write()** dodaje int broj u bafer, dok metod **read()** vraća int broj iz bafera.

12. Kada pozivate read() metod a red je prazan, šta se dešava?

- Kada zadataka čita int vrednost iz bafera i ako je bafer prazan, tada zadatak mora da čeka za **notEmpty** uslov da bude ispunjen.

13. Šta se dešava kada pozivate wite() metod a red je pun?

- Kada zadatak doda vrednost u bafer, ako je bafer pun zadatak će čekati uslov **notFull** da bude ispunjen.

14. Šta je blokirajući red? Koji su blokirajući redovi podržani u Javi?

- Blokirajući red je red koji čini da nit blokira unos elemenata u puni red ili da blokira uklanjanje elementa iz praznog reda.
Tipovi blokirajućih redova su:
  - ArrayBlockingQueve,
  - LinkedBlockingQueve,
  - PriorityBlockingQueve.

15. Koji metod koristite za dodavanje elemenata u ArrayBlockingQueve? Šta se dešava kada je red već pun?

- Za dodavanje elemenata u BlockingQueve koristi se metod **put()**. Kada je red već pun, metod put() čeka, tj. prelazi u stanje čekanja dok se ne oslobodi red za dodavanje novih elemenata.

16. Koji metod koristite da dobijete i izbacite element iz blokirajućeg reda tipa ArrayBlockingQueve? Šta se dešava ako je red već prazan?

- Za dobijanje i uklanjanje elementa iz blokirajućeg reda koristi se metod **take()**. Ukoliko je red već prazan, metod take() čeka, tj. prelazi u stanje čekanja dok se ne pojave novi elementi u redu koji bi se mogli ukloniti.

17. Kako definišete klasu zadatka? Kako kreirate nit za neki zadatak?

- Zadaci su objekti. Klasa koja predstavlja zadatke mora da primenjuje interfejs Runnable. Ona se definiše u sledećem obliku:

public class TaskClass implements Runnable { … }

Nit za neki zadatak se kreira pomoću konsturktora koji kreira objekat niti kao:

Threead nit = new Thread(task);

pri čemu je task zadatak, tj. objekat klase TaskClass.

18. Šta bi se desilo ako bi zamenili metod start() sa metodom run() u linijama 14-16 listinga klase TaskTreadDemo?

- Ništa, program bi isto radio, zato što pozivom metoda start() svake od niti počinje izvršenje niti, odnosno izvršavaju se odgovarajuće run() metode za svaki zadatak.

19. Koje su sličnosti i razlike između ključa i semafora?

- **Zaključavanje** omogućava da samo jedna nit uđe u deo koji je zaključan i zaključavanje se ne deli sa bilo kojim drugim procesom.

Semafor omogućava deljenje sa drugim procesima ali ograničava njihov broj.

20. Kako kreirate semafor koji dozvoljava korišćenje tri simultane niti? Kako zahtevate dozvolu od semafora? Kako vraćate dozvolu semaforu?

- Semafor sa dozvolama za korišćenje 3 simultane niti se kreira kao:

Semaphore semafor = new Semaphore(3);

Nit zahteva dozvolu pozivajući metod acquire() klase Semaphore:

semafor.acquire();

Nit vraća (oslobađa) dozvolu pozivanjem metoda release() klase Semaphore:

semafor.release();

21. Šta je stanje niti? Objasni stanja niti?

- Stanje niti ukazuje na status niti. Zadaci se izvršavaju u nitima a niti kao objekti za izvršenje zadataka mogu biti u jednom od pet stanja:

1) New – je početno stanje kod novokreiranih niti,
2) Ready – posle početnog pod dejstvom metode start() nit ulazi u ovo stanje,
3) Running – kad OS alocira vreme CPU-a da počne izvršenje niti i to izvršenje zaista i počne, dobija se stanje Running,
4) Blocked – je stanje kada nit postane neaktivna, ukoliko se pozovu metodi join(), sleep(), wait().
5) Finished – nit je u ovom stanju ako je u celosti izvršila svoj zadatak definisan metodom run().

22. Šta je sinhornizovana kolekcija? Da li je ArrayList sinhronizovana? Kako je možete sinhronizovati?

- Sinhornizovana kolekcija služi za zaštitu podataka u kolekciji. One se mogu koristiti za liste, setove, mape.
ArrayList nije sinhronizovana i nju možemo sinhronizovati primenom metode synchronizedList() koja vraća sinhronizovanu listu iz specijalizovane liste.

23. Objasnite zašto iterator brzo pada u uslovima višenitnosti?

- Kada iterator prelazi po kolekciji koju istovremeno menja neka druga nit, onda iterator odmah pada izbacujući CrrentModificationException, koji je potklasa klase RuntimeException. Da bi se ovo izbeglo, potrebno je kreirati sinhronizovani objekat kolekcije i zahtevati ključ za taj objekat kada po njemu prelazimo.

24. Kako definišete ForkJoinTask? Koje su razlike između klasa RecursiveAction i RecursiveTask?

- **ForkJoinTask** je apstraktna klasa za sve zadatke.
Razlika je u tome što:

- Klasa RecursiveAction definiše kako zadatak treba da se izvrši.
- Klasa RecursiveTask definiše kako će zadatak da se izvrši i vraća vrednost posle završetka zadatka.

25. Kako saopštavate sistemu da izvrši neki zadatak?

- Metodi **invoke()** i **invokeAll()** implicitno pozivaju metod **fork()** da izvrši zadatak i metod **join()** da sačeka završetak svih zadataka i da vrati rezultat.

26. Koji metod upotrebljavate za testiranje da li je neki zadatak završen?

- Koristi se metod **isDone()** koji vraća True ako je zadatak završen.

27. Kako kreirate ForkJoinPool? Kako ubacujete zadatak u FrkJoinPool?

- Metoda **ForkJoinPool()** kreira ForkJoinPool sa svim raspoloživim procesorima.
  Metoda **invoke()** ubacuje i izvršava zadatak i vraća rezultat posle završetka.

28. Za simultani rad sa više klijenata nije potrebno kreirati po jednu nit za svakog kijenta, da ili ne? ***

- False.

29. Šta od navedenog je tačno? ***

- Niži prioritet se koristi samo kada više nema niti sa višim prioritetom.
- Ako sve niti imaju isti prioritet, da bi se izbegla blokada, onda svi dobijaju jednako CPU vreme.
- Niži prioritet se koristi samo kada više nema niti sa višim prioritetom.

30. Da bi kreirali semafor, potrebno je da specificirate broj dozvola koje može da izda (pa samim tim i broj niti koje mogu simultano pristupiti deljivom resursu) kao i fer postupak koji se primenjuje pri dodeli dozvola? Da ili ne? ***

- True.

31. Metoda sleep() može da izbaci izuzetak InterruptedException, koji spada u kategoriju proverenih izuzetaka? ***

- True

32. Koje Java klase koristimo za blokirajuće redove? ***

- ArrayBlockingQueve
- LinkedBlckingQueve
- PriorityBlockingQueve

TEST 2

1. Kako dobijate objekat klase InetAddress?

- Objekat klase InetAddress dobijamo pomoću iskaza:
InetAddress inetAddress = socket.getInetAddress();

2. Koje metode koristite da bi dobili IP adrese i naziv host računara iz InetAdress?

- IP adresu dobijamo pomoću: **getHostAddress()** metode.
Naziv računara dobijamo pomoću: **getHostName()** metode.

3. Šta će se desiti ako nije u liniji 227 llistinga klase TicTacToeClient definisana veličina ćelije?

- Ne znam ??????????????????????????????

4. Ako igrač nije na potezu a klikne na praznom polju, šta će klijentski program dat klasom TicTacToeClient uraditi?

- Neće uraditi ništa, već će čekati na potez pravog igrača, zato što po uspostavljanju sesije, server naizmenično prima poteze od igrača i po prijemu poteza od igrača server određuje status igre. Ako igra nije završena, server šalje status continue i šalje potez igrača drugom igraču.

5. Šta od navedenog je tačno? ***

- Kada je veza uspostavljena, klijent i server komuniciraju preko softverskih utičnica, tj. objekata klase ServerSocket i Socket.
- Utičnice su krajnje tačke logičkih veza između dva računara i koriste se za slanje i primanje podataka.

6. Kada server prihvati vezu, komunikacija između servera i klijenta se vodi na isti način kao i kod U/I tokova podataka (strimova)? ***

- True

7. Server koristi ime definisano promenljivom serverName, čija vrednost je ili Internet ime domen servera ili njegova IP adresa, da ili ne? ***

- True

8. Šta od navedenog je tačno? ***

- TCP omogućava da dva servera uspostave vezu i da razmene tokove podataka.
- IP adresa je sačinjena od četiri broja od 0 do 255 između kojih se nalaze tačke.
- IP je adresa koja na jedinstveni način utvrđuje računar na internetu.

9. Da li je moguće da se klijent i server izvršavaju na različitim računarima a da to ne ometa njihovu komunikaciju? ***

- Da

TEST 3

1. Opišite unapred pripremljene iskaze. Kako kreirate primerak klase PreparedStatement? Kako izvršavate PreparedStatement? Kako unosite vrednosti parametara u PreparedStatement objekat?

- PreparedStatement omogućava kreiranje parametrizovane SQL izjave. Kreiramo ga sa:
  PrepraredStatement preparedStatement = connection.prepareStatement("insert into Student (firstName, mi, lastName)" + "values(?,?,?)");
- Izvršenje se vrši pozivom **executeQuery()**.
- Unosimo ih pomoću metode **setX(int parameterIndex, X value)**; Na primer:
  preparedStatement.setString(1, "A");

2. Koje su povoljnosti koje pruža primena unapred pripremljenih iskaza?

- Prednost je u tome što PreparedStatement omogućava predkompajliranje SQL iskaza sa ili bez parametara, čime su ovi iskazi efikasniji za ponovna izvršenja.

3. Za šta služi DatabaseMetaData? Opišite metode DatabaseMetaData. Kako dobijate primerak od DatabaseMetaData?

- DatabaseMetaData je interfejs koji suži za dobijanje informacija o bazi podataka. Za dobijanje primerka interfejsa DatabaseMetaData za bazu podataka koristi se metoda **getMetaData().**

4. Za šta služi ResultSetMetaData? Opišite metode u ResultSetMetaData? Kako dobijate primerak ResultSetMetaData?

- ResultSetMetaData je interfejs koji služi za dobijanje informacija u specificiranom ResultSet. Za dobijanje primerka ResultSetMetaData, koristi se metoda **getMetaData()** na skupu rezultata.

5. Kako nalazite broj kolona u skupu rezultata? Kako nalazite nazive kolone u skupu rezultata?

- Broj kolona u skupu rezultata nalazimo pomoću metode **getColumnCount()**.
- Nazive kolona u skupu rezultata nalazimo pomoću metode **getColumnName()**.

6. Koje su prednosti razvoja aplikacije sa bazom podataka upotrebom Jave?

- Prednost je u tome što ove aplikacije mogu da koriste više sistema baza podataka koji su u mreži računara koje koriste.

7. Opišite sledeće JDBC interfejse: Driver, Connection, Statement i ResultSet?

- **Driver** – je uslužni program
- **Connection** – omogućava povezivanje sa bazom podataka.
- **Statement** – kreira SQL iskaz.
- **ResultSet** – obrađuje rezultate.

8. Kako instalirate JDBC drajver? Koje su drajver klase za MySQL, Access i Oracle?

- JDBC drajver instaliramo pomoću: **Class.forName("com.mysql.jdbc.Driver");**
Drajver klase su:
Access: **sun.jdbc.odbc.jdbcOdbcDriv**
MySQL: **com.mysql.jdbc.Driver**
Oracle: **oracle:jdbc.driver.OracleDriver**

9. Kako kreirate vezu sa bazom podataka? Koje se URL koriste za MySQL, Access i Oracle?

- Povezivanje baze podataka se vrši pozivanjem statičkog metoda **getConnection()** u klasi DriverManager.
URL za razne DBMS su:
MySQL: **jdbc:mysql://hostname/dbname**
Access: **jdbc:odbc:dataSource**
Oracle: **jdbc:oracle:thin:@hostname:port#:oracleDBSID**

10. Kako kreirate Statement i izvršavate SQL iskaz?

- Statement objekat definiše SQL islaze koje DBMS treba da izvrši i da vrati dobijeni rezultat. On izgleda ovako: **Statement statement = connection.createStatement();**
SQL upit se može izvršiti upotrebom **executeQuery(String sql)**, a rezultat se vraća u obliku SQL objekta.

11. Kako dobijate vrednosti u ResultSet?

- Za dobijanje vrednosti u ResultSet koristimo metod **getMetaData()**.

12. Šta su superključevi, ključevi kandidati i primarni ključevi?

- **Superključ** je atribut ili skup atributa koji na jedinstveni način identifikuje jednu relaciju. Skup svih atributa u jednoj relaciji je superključ.
- **Ključ-kandidat** je podskup superključa koji može da ima više ključa-kandidata.
- **Primarni ključ** je jedan od ključeva-kandidata određen od strane projektanta baze i često se koristi za identifikaciju slogova table.

13. Šta je sekundarni (strani) ključ?

- Sekundarni (strani) ključ ograničava veze između relacija.

14. Da li neka relacija ima više od jednog primarnog ključa u istoj relaciji?

- Ne

15. Da li strani ključ mora da ima isto ime kao što je pozvani primarni ključ?

- Ne

16. Da li strani ključ može da ima vrednost null?

- Ne

17. Izbaciti pojmove koji ne pripadaju skupu? ***

- Set i HashSet.

18. JDBC je Java API koji se koristi za razvoj aplikacija koje koriste relacione baze podataka? ***

- True

19. Čemu služi interfejs Connection? ***

- Za povezivanje sa bazom podataka.

20. Kako se naziva univerzalni jezik za pristupanje relacionim sistemima baza podataka? ***

- SQL

21. Šta je RDBMS? ***

- Sistem za upravljanje bazom podataka.

TEST 4

1. Šta su DAO objekti?

- DAO objekti su objekti koji omogućavaju pristup bazi podataka unutar DAO klase.

2. Zašto primena DAO šablona predstavlja dobru praksu za pisanje koda koji se obraća bazi podataka?

- Zato što se njihovom primenom postiže jasno razdvajanje modula programa pomoću kojeg ostali slojevi aplikacije, poput logike korisničkog interfejsa ili poslovne logike ne zavise od logike perzistencije.

3. Šta je obeleženi upit?

- To je poseban specificirani upit koji sadrži listu od više upita.

4. Šta je JPQL?

- JPQL je JPA specifični jezik upita, čija je sintaksa veoma slična SQL jeziku.

5. Navedite i objasnite anotacije za proveru podataka?

- **@NotNull** – ne dozvoljava polju da prihvati null vrednost.
- **@Size** – ukazuje na maksimalan broj karaktera kojih osobina zrna moze da prihvati.
- **@Pattern** – obezbedjuje da se obelezeno polje podudara sa regularnim iskazom.

6. Navedite i objasnite kardinalnosti relacija JPA entiteta?

- @OneToMany ← Ovde jos napisi objasnjenje za svaki
- @JoinCoulumn
- @ManuyToOne
- @ManyToMany
- @OneToMany

7. Pokazati na primeru kardinalnosti JPA entiteta?

- @ManyToMany, ovde je primer slučaj sa porudžbinama, gde jedna porudžbina može da sadrži više proizvoda i jedan proizvod može da bude sadržan u više porudžbina.

8. Objasnite proceduru automatskog generisanja JSF stranica iz JPA entiteta?

- Koraci su:
  1) Klik na File → New File → odaberemo kategoriju datoteke (Java Server Faces ili Persistance) → odaberemo tip datoteke (JSF Pages from Entity Clases).
  2) Klik na Next → markiraju se klase → klik na Add → klik na Next.

  Prolaskom kroz zadati Wizard, kreiraju se JSF stranice za sve klase entiteta, koje su bile izabrane.

9. Objasnite proceduru kreiranja JPA entiteta iz tabela baze podataka?

- ??????????????????????????????

10. Objasnite anotacije koje koriste JPA entiteti?

- Java klasa koja se odnosi na JPA entitet je obelezena anotacijom @Entity ?????????????

11. Šta su JPA entiteti?

- JPA entiteti su Java klase čija se polja čuvaju (perzistiraju) u bazama podataka pomoću JPA API.

12. Kojom anotacijom se obeležavaju JPA entiteti?

- @Entity

13. Šta je **connection pool** i koje su mu prednosti?

- Informacije kao što su: naziv servera, porta, kredencijala i slično, koje omogućavaju konektovanje na bazu podataka, nazivaju se connection pool.
Pomoću connection pool-a se povećavaju performanse aplikacije zahvaljujući tome što se izbegavaju zahtevne akcije otvaranja i zatvaranja konekcije.

14. JPQL je JPA specifični jezik upita čija je sintaksa veoma slična SQL jeziku? ***

- True

15. Izbaci pojmove koji ne pripadaju navedenim? ***

- JDBC (a pripadaju: EclipseLink, Hibernate, TopLink, Essentials, iOpenJPA, KODO)

16. Da li JPA dozvoljava automatsko generisanje primarnih ključeva? ***

- Da primenom anotacije @GeneratedValue u kombinaciji sa anotacijom @Id.

17. EclipseLink je podrazumevana JPA implementacija za GlassFish aplikativni server, da ili ne? ***

- True

18. DAO šabloni dizajna čuvaju sve funkcionalnosti pristupa bazi podataka unutar DAO klase? ***

- True

TEST 5

1. Šta je HQL?

- **HQL** (Hibernate Quality Language) je objektno-orjentisan jezik za rad sa bazom podataka u okviru Hibernate ORM. On radi sa trajnim objektima i njihovim svojstvima. HQL upiti se prevode u SQL upite koji izvršavaju potrebne akcije na bazi podataka.

2. Koji značaj ima objekat Criteria?

- Objekat **Criteria** omougćava da definišemo kriterijume za programirani odabir objekata primenom pravila filtriranja i logičkih uslova.

3. Za šta je moguće iskoristiti standardne SQL upite pored HQL upita?

- Moguće je iskoristiti za dobijanje specifičnh svojstava baze podataka kao što je korišćenje CONNECT ključne reči.
  Hibernate 3.X dozvoljava primenu SQL upita, memorisanih procedura, operacija create, update, delete i load za sve slogove.
  Aplikacija će kreirati originalne SQL upite iz Session objekta primenom metoda **createSQLQuery()**.
  Moguće je povezati dobiveni SQL rezultat sa postojećim HQL rezultatom, primenom metoda: **addEntity()**, **addJoin()**, **addScalar()**.

4. Koje su prednosti Hibernate anotacija?

- **Hibernate Annotations** je moćan način obezbeđivanja meta podataka za mapiranje objekata u relacione tabele. Svi meta podaci se stavljaju zajedno u POJO java fajl zajedno sa programskim kodom, pa korisnik odatle razume strukturu tabela I simultano vreme razvoja.

5. Objasnite najčešće korišćene anotacije?

- Najčešće korišćene anotacije su:
  @Entity – njome se obeležavaju JPA entiteti
  @Table – dozvoljava specificiranje detalja tabele koja će se koristiti za trajno memorisanje u bazi podataka.
  @Id – omogućava pristup svojstvima nekog objekta za vreme izvršenja programa.
  @GeneratedValue – omogućava promenu vrednosti primarnog ključa. Ima 2 parametra: Strategy i Generator.
  Ove anotacije sve zajedno daju odgovarajuće instrukcije mapiranja programu tokom njegovog izvršavanja.

6. Zbog čega koristiti keširanje?

- **Keširanje** optimizuje performanse aplikacije i funkcionalno se smešta između aplikacije i baze podataka sa ciljem da se minimalizuje komunikacija sa bazom podataka i na taj način poboljšaju performase (brzina rada) aplikacije.

7. Kako funkcioniše keširanje kod Hibernate?

- Keširanje kod Hibernate-a upotrebljava šemu keširanja u više nivoa:
  a) Prvi nivo keša je **keš Session objekta** i ovo je obavezni keš kroz koji moraju da prođu svi zahtevi.
  b) Drugi nivo keša je **Opcioni keš** i on se konfiguriše prema klasi ili prema kolekciji i najviše se koristi u slučaja rada sa više sesija (tj. Session objekata). Na ovom nivou postoje 4 strategije konkurentnog rada: transactional, read-write, nonstrict-read-write i read-only.
  c) Treći je keš na nivou upita, koji se koristi najviše kod upita koji često ponavljaju iste parametre.

8. Šta su presretači i koja ima je Namena?

- **Presretač interfejs** (Interceptor) obezbeđuje metode koji se pozivaju u različitim stanjima objekta da bi obavili zahtevane zadatke. Njegovi metodi omogućavaju aplikaciji da proverava sadržaj i manipuliše svojstvima trajnog objekta pre nego što se objekat uskladišti ponovo, promeni, izbriše ili dovede u memoriju.

9. Šta je ORM?

- ORM (Object Relational Mapping) predstavlja tehniku programiranja kojom se vrši konverzija podataka između relacionih baza podataka i OOP jezika kao što su: Java, C# i sl. ORM je i radni okvir (framwork) za mapiranje objktno-orjentisanog Domain modela na tradicionalnim relacionim bazama.

10. Koje su prednosti ORM u odnosu na JDBC?

- ORM obezbeđuje sledeće prednosti u odnosu na JDBC:
  * Omogućava programu da pristupa objektimma a ne tabelama u bazi,
  * Sakriva detalje o SQL upitimma od OO programa i njegove logike,
  * Baziran je na JDBC koji je u pozadini,
  * Nezavistan je od implementacije baze podataka, tj. proizvođača DBMS,
  * Primenjeni entiteti su prilagođeni poslovnom konceptu, a ne strukturi baze podataka,
  * Upravljanje transakcijama i automatsko generisanje ključeva,
  * Brz razvoj aplikacija, … i sl.

11. Koji od navedenih objekata predstavlja objekat koji se koristi za kreiranje i izvršenje OO kriterijuma upita za prikupljanje objekata?***

- Criteria

12. Šta je Hibernate?

- **Hibernate** je softverska komponenta koja se nalazi između Java objekata (u aplikaciji) i server sa bazom podataka i koja obavlja celokupan posao vezan za mapiranje objekata u relacione tabele.

13. Objasnite ulogu Configuration, Connection i Mapping?

- **Configuration** objekat kreira SessionFactory objekat koji konfiguriše Hibernate aplikaciju pri čemu koristi konfiguracioni fajl i dozvoljava kreiranje Session objekata.
- **Connection** komponentu obezbeđuje jedan ili više konfiguracionih fajlova koji podržavaju Hibernate.
- **Mapping** kreira vezu između Java Klasa i tabele baze objekata.

14. Objasnite objekte SessionFactory, Session, Transaction, Query i Criteria?

- **SessionFactory** je objekat koji konfiguriše Hibernate za aplikaciju pri čemu koristi konfiguracioni fajl i dozvoljava kreiranje Session objekata.
- **Session** objekat se koristi za dobijanje fizičke veze sa bazom podataka.
- **Transaction** objekat predstavlja jedinicu rada sa bazom podataka.
- **Query** objekat se koristi za povezivanje parametara upita, ograničenje broja rezultata koje vraća upit i na kraju izvršava upit.
- **Criteria** objekat se koristi za kreiranje i izvršenja OO kriterijuma upita za prikazivanje objekata.

15. Kako se Hibernate konfiguriše?

- Hibernate se **konfiguriše** određivanjem vrednosti njegovih svojstava. Svojstva se definišu za slučaj korišćenja baze podatka ili JNDI aplikacionog server.

16. Objekat koje klase konfiguriše Hibernate za aplikaciju pri čemu koristi konfiguracioni fajl i dozvoljava kreiranje Session objekata? ***

- SessionFactory.

17. SessionFactory objekat kreira Configuration objekat koji pak konfiguriše Hibernate za aplikaciju pri čemu koristi konfiguracioni fajl i dozvoljava kreiranje Session objekata? ***

- False (zapravo je obrnuto)

18. Rešenje za probleme neusklađenosti objektnog i relacionog modela predstavlja? ***

- ORM

19. Objekat koje klase se koristi za dobijanje fizičke veze sa bazom podataka? ***

- Session