1. Šta je set? Koje vrste setova poznajete?

Interfejs Set proširuje interfejs Collection. On obezbeđuje da Set primerak ne sadrži duplirane elemente. Konkretne klase koje primenjuju interfejs Set moraju da obezbede da se ne mogu dodati duplikati elemenata koji su već u setu. Ovo znači da dva elementa e1 i e2 u jednom setu ne mogu da daju: true (istinito) u logičkom izrazu: **e1.equals(e2).**

Postoje tri vrste seta: HashSet, LinkedHashSet I TreeSet.

1. U čemu je suštinska razlika seta i liste?

Set ne dozvoljava ponavljanje elemenata, tj duplikate, a lista do dozvoljava.

1. Šta je heširanje?

Heširanje(hashing) je tehnika koja nalazi vrednost u nekom nizu upotrebom indeksa koji se dobija iz ključa, bez sprovođenja pretraživanja.

1. Diskutujte performanse setova i listi?

Setovi su mnogo efikasniji od listi kod testa da li je neki broj u setu ili listi. Razlog je u implementacijama listi i setova.

1. Šta je mapa? Šta omogućava interfejs Map?

Mapa je kontejner objekat koji skladišti kolekciju parova ključ/vrednost . Ona omogućava brzu pretragu, brisanje i menjanje parova primenom ključa pretrage. Mapa skladišti vrednosti zajedno sa ključevima.

Ključevi su kao indeksi. U listi (List), indeksi su celi brojevi. U mapi (Map) ključevi mogu biti bilo kakvi objekti. Mapa ne može da sadrži duple ključeve, tj. dva ista ključa. Svaki ključ odgovara jednoj vrednosti.

1. Koje vrste mapa poznajete? Kako se kreiraju objekti mapa?

Postoji tri tipa mapa: heš mapa (HashMap), povezana keš mapa (LinkedHashMap) i mapa stabla (TreeMap).

Klasa HashMap je efikasna za lociranje neke vrednosti, ubacivanje ulaza i za brisanje ulaza. Klasa LinkedHashMap proširuje HashMap sa implementacijom povezane liste koja podržava redosled ulaza u mapi. Ulazi u HashMap nisu uređeni po redosledu , ali ulazi u LinkedHashMap mogu se dobijati ili po redosledu u kom su uneti u mapu, ili po redosledu po kom su poslednji put bili imali pristup.

Klasa TreeMap je efikasna u pretraživanju sortiranih ključeva. Ključevi se mogu sortirati upotrebmo interfejsaComparable ili Comparator. Ako kreirate objekat TreeMap upotrebom konstruktora bez argumenata, metod compareTo() u interfejsu Comparable upoređuje ključeve u mapi, pretpostavljajući da klasa za ključeve primenjuje interfejsComparable.

Map<String, Integer> mapa = new HashMap<>();

1. Šta je nit, a šta zadatak?

Nit obezbeđuje mehanizam za izvršenje jednog zadatka. U Javi se mogu istovremeno aktivirati više niti u jednom programu radi istovremenog izvršenja više zadataka, ako računarski sistem koristi više procesora. Međutim, kod jednoprocesorkih sistema, više niti koriste isti procesor. To tzv. deljenje vremena (time sharing) procesora po zadacima obezbeđuje operativni sistem, koji vrši raspodelu resursa sistema istovremenim korisnicima sistema.

1. Kako se kreira nit, a kako zadatak?

Klasa Task mora implementirati interfejs Runnable I da override-uje metodu run();

Task task = new Task();

Thread thread = new Thread(task);

thread.start();

1. U čemu se razlikuju pozivi metoda run() i start()?

Metoda run() se implementira iz interfejsa Runnable I u njoj definešemo kako želimo da se naš zadatak ponaša. Kada kreiramo nit sa zadatkom, startujemo tu nit metodom start();

1. Objasnite primenu metoda klase Tread za upravljanje izvršavanjem niti?

yield() da bi privremeno oslobodili vreme procesora za druge niti.

sleep(long millis) stavlja nit u stanje spavanja određeni broj milisekundi da bi dozvolio drugim nitima da se izvršavaju. Sleep se stavlja u trycatch blok.

join() da bi obezbedili da jedna nit čeka da druga nit završi.

setPriority() se određuje prioritet niti, a metodom getPriority() se dobija prioritet niti. Klasa Thread ima sledeće int kontante: MIN\_PRIORITY, NORM\_PRIORITY, i MAX\_PRIORITY, koje određuju prioritet 1, 5 I Prioritet glavne niti (main) je Thread.NORM\_PRIORITY.

1. Šta je pul niti?

Upotreba pula niti je idealan način za istovremeni rad sa puno zadataka. Java obezbeđuje interfejs Executor za izvršenje zadataka u jednom pulu niti i interfejs ExecutorService za upravljanje i kontrolu zadataka. Ako imamo veliki broj zadataka, kreiranje novog objekta Thread za svaki zadatak je neefikasno.

1. Po čemu se razlikuju interfejsi Executor i ExecutorService?

interfejs Executor za izvršenje zadataka u jednom pulu niti i interfejs ExecutorService za upravljanje i kontrolu zadataka.

1. Šta je sinhronizacija niti i zašto je potrebno obaviti sinhronizaciju?

Sinhronizacija niti je koordinacija izvršenja zavisnih niti. Deljeni resursi, koje koriste više niti, mogu biti oštećeni ako su izloženi simultanom pristupu više niti. Na primer ako u isto vreme više niti pokušava da pristupi bazi, da na primer promeni stanje na bankovnom računu.

Konflikt između dve niti koje istovremenu pristupaju zajedničkom resursu je poznat kao race condition (uslov takmičenja). Klasa je bezbedna za nit (thread-safe) ako ne dovodi do stvaranja race condition-a.

1. Kada koristimo rezervisanu reč synchronized?

Da bi se izbegli race condition, treba preventivno sprečiti da više od jedne niti simultano uđe u određeni deo programa, koji se naziva kritičnim regionom. Možete iskoristiti ključnu reč da bi sinhronizovali metod tako da samo jedna nit može da pristupi resursu vezanom za tu ključnu reč u nekom vremenskom trenutku.

Sinhronizovani metod zahteva zaključavanje (lock) pre izvršenja. Zaključavanje je mehanizam za ekskluzivno korišćenje resursa. U slučaju objektnog metoda, vrši se zaključavanje objekta za koji se metod poziva. U slučaju statičkog metoda, zaključava se klasa.

Ako jedna nit poziva sinhronizovani objektni metod (metod primerka), prvo treba zaklučati taj objekat, a onda pustiti izvršenje metoda, a posle toka otključati objekat. Ako neka druga nit poziva istu objektnu metodu, zbog njegove privremene blokade, moraće da sačeka deblokiranje (otključavanje) metoda.

1. Na koje načine je moguće obaviti sinhronizaciju niti?

Pomoću ključne reči synchronized I upotrebom ključeva. Sinhronizovan objektni metod implicitno zahteva ključ (otvaranje) objekta (primerka) pre izvršenja metoda. Java vam omogućava da eksplicitno zahtevate ključ, što vam daje više kontrole traženja i oslobađanja ključeva.

Interfejs Lock koristi metod newCondition() za kreiranje bilo kog broja Condition objekata, koji mogu da budu upotrebljeni za komunikacije sa nitima.

ReentrantLock je konkretna primena Lock interfejsa radi kreiranja višestruko ekskluzivnih ključeva. Možete kreirati ključ sa specificiranom politikom fer ponašanja (fairness policy). Ispravna (true) politika fer ponašanja garantuje da nit koja najduže čeka, prva dobija kljuć.

1. U čemu se razlikuju pozivi metoda newFixedThreadPool(int) i newCachedThreadPool()?

**newFixedThreadPool(int)** kreira fiksni broj niti u pulu. Ako nit završi izvršenje zadatka, može se ponovo koristiti za neki drugi zadatak. Ako nit završi zbog problema pre zatvaranja, kreira se nova nit da bi ga zamenila ako sve niti u pulu nisu van rada i ako ima zadataka koji čekaju na izvršenje.

**newCachedThreadPool(**) kreira novu nit ako nisu sve niti u pulu vana rada i ako ima zadataka koji čekaju na izvršenje. Nit u keš pulu će se završiti ako nije u radu 60 sekundi. Keš pul je efikasan kada ima puno kratkih zadataka.

1. Objasnite sinhronizaciju upotrebom ključeva?

Kreiramo instacu interfejsa Lock, tj. njegovu konkrentnu primenu ReentrantLock; U metodi koju želimo da sinhronizujemo, na samom početku zaključavamo, pozivom metode **lock()**, a nakon što se dođe do kraja metode, otključavamo metodom **unlock();**

1. Šta predstavlja i kako se realizuje saradnja (kooperacija) između niti?

Kooperacija između niti služi da jedna nit čeka drugu kako bi izvršila zadatak, jer je možda neophodno da ta druga nit izvrši neki zadatak pre nego što prva nit može da nastavi sa izvršenjem.

Saradnja se realizuje korišćenjem klase **Condition uslov = lock.newCondition();** Kada se kreira uslov, onda metodama await(), signal() I signalAll vršimo kooperaciju među njima.

1. Šta je blokada ključeva i kako se ona sprečava?

Ponekad, dve ili više niti zahtevaju ključeve za pristup nekoliko deljivih objekata, tj. objekata koje zajednički koriste. To može da dovede do zastoja (deadlock) u kome svaka nit ima ključ za jedan od objekata i čeka ključ za neki drugi objekat.

Blokada ključeva se lako izbegava upotrebom jednostavne tehnike poznate kao redosled resursa (resource oredering). Sa ovom tehnikom, vi određujete redosled svih objekata čiji se ključevi traže da bi obezbedili da svaka nit dobija ključ po tom redosledu.

1. Šta je blokirajući red i šta je njegova namena?

Blokirajući red je red koji čini da nit blokira unos elementa u puni red ili da blokira uklanjanje elementa iz praznog reda.

Interfejs BlockingQueue proširuje java.util.Queue i obezbeđuje sinhronizovane metote put() i take() za dodavanje elementa na kraj reda i za uklanjanje elementa sa počeka (vrha) reda

1. Koje tipove blokirajućih redova poznajete?

Za podršku blokirajućih redova, Java obezbeđuje tri klase za tri tipa blokirajućih redova: **ArrayBlockingQueue, LinkedBlockingQueue, I PriorityBlockingQueue**

Ove klase se nalaze u paketu java.util.concurrent.

**ArrayBlockingQueue** je tip blokirajućeg reda koji upotrebljava niz (array). Morate da specificirate njegova kapacitet ili opciono njegov fer postupak, da bi se konstruisao onjekat ove klase.

**LinkedBlockingQueue** je tip blokirajućeg reda koji upotrebljava povezane liste (linked list). Mogu se kreirati ogranični ili neograničeni blokirajući redovi ovog tipa.

**PriorityBlockingQueue** je tip blokirajjućeg reda koji upotrebljava prioritetni red (priority queue). Mogu se kreirati ogranični ili neograničeni blokirajući redovi ovog tipa

1. Šta je semafor i šta je njegova namena?

U računarskim naukama, semafor je objekat koji kontroliše pristup zajedničkim resursima. Pre nego što pristupi resursu, nit mora da zatraži dozvolu od semafora. Kada završi korišćenje resursa, nit mora da vrati dozvolu semaforu.

Semafori se koriste za ograničenje broja niti koje mogu da pristupe deljivom resursu.

1. Koja stanja nit može da ima?

Niti mogu biti u jednom od pet stanja: New, Ready, Running, Blocked, ili Finished.

Kod novokreiranih niti, početno stanje je New. Posle početka rada niti dejstvom metoda start(), niz ulazi u stanje Ready. U tom stanju nit je spremna za rad (runnable), ali još ne radi. Neophodno je da operativni sistem alocira vreme procesora (CPU) za račeka otvaranjed niti. Kada spremna nit počne izvršenje, dobija staus Running. Iz stanja Running, nit se može vratiti u stanjeReady, ako dato vreme CPU istekne ili je pozvan metod yield().

Nit može da uđe i u status Blocked (kada postaje neaktivna) iz nekoliko razloga: pozvani su metodi join(), sleep(), ili wait(), ili čeka završetak I/O operacije. Blokirana nit se reaktivira dejstvom obrnute akcije onoj koja je dovela do blokade.

Na primer, isteklo je vreme trajanja metoda sleep(), te se nit reaktivira i ulazi u stanje Ready. Na kraju, nit je u stanjuFinished, ako je u celosti izvršila svoj zadatak definisan metodom run(). Metod isAlive() se koristi da se nađe stanje niti. Vraća true ako je nit u stanju Ready, Blocked ili Running. Vraća false, ako je nit nova i još nije startovana, ili ako je u stanjuFinished.

1. Kako se i iz kojeg razloga sinhronizuju kolekcije?

Klase u Java Collections Framework nisu nitno-bezbedne, tj. njihov sadržaj može da bude oštećen ako im se pristupa istovremeno od strane više niti. Možete zaštiti podatke u kolekciji zaključavanjem kolekcije ili upotrebom sinhronizovanih kolekcija.

Klasa Collections obezbeđuje šest statičkih metoda za umotavanje kolekcije u sinhronizovanu verziju . Kolekcije kreirane ovim metodima se nazivaju sinhronizovani omotači (synchronization wrappers).

Metod synchronizedCollection(Collection c) vraća novi Collection objekat u kome su svi metodi koji pristupaju originalnoj kolekciji sinhronizovani

Sinhronizovane klase omotači su nitno-bezbedne, ali iteratori brzo padaju. To znači da kada interator prelazi po kolekcije koju istovremeno menja neki druga nit, onda interator odmah pada izbacujući java.util.ConcurrentModificationException, koji je potklasa klase RuntimeException. Da bi se ovo izbeglo, potrebno je da kreirate sinhronizovni objekat kolekcije i da zahtevate ključ za taj objekat kada po njemu prelazite.

1. Šta je paralelno programiranje?

Paralelno programiranje je tehnika programiranja koja omogućava korišćenje više niti koji se paralelno izvršavaju rešavajući nezavisne potprobleme problema za koji se program pravi.

Pojava sisema sa više procesora i sa više jezgara stvorili su revoluciju u razvoju softvera. Da bi se iskoristile mogućnosti koje pružaju višeprocesorki sistemi, potrebno je razviti softvere koji se izvršavaju paralelno na više procesora. JDK 7 je za ovu namenu obezbedio Fork/Join Framework za paralelno programiranje, koji koristi procesore sa više jezgara (multicore processors).

Problem je podeljen na dva nezavisna potproblema koji se ne preklapaju, i koji se zbog toga mogu da rešavaju nezavisno. To je primena koncepta “podeli-i-osvoji” (devide-and-conquer) u oblasti paralelnog programiraja. U JDK 7 Fork/Join Framework, “viljuška” , tj. račva se posmatra kao nezavistan zadatak koji izvršava u svojoj niti.

1. Šta je ForkJoinTask, a šta ForkJoinPool?

Radni okvir Fork/Join Framework definiše zadatak upotrebom klase ForkJoinTas.

ForkJoinTask je apstraktna klasa za sve zadatke. Ona liči kao na nit, ali je mnogo “lakša” jer se veliki broj zadataka i podzadataka može izvršiti sa manjim brojem stvarnih niti koje definiše klasa ForkJoinPool. Zadaci su prvenstveno koordinisani upotrebom metoda **fork()** i **join().** Pozivm metoda **fork()** za jedan zadatak priprema se asinhrono izvršenje, a pozivom metodajoin(), čeka se završetak zadatka. Metodi **invoke()** i **invokeAll()** implicitno pozivaju metod **fork()** da izvrši zadatak i metod **join()** – da sačeka završetak svih zadataka i da vrati rezultat. Statički metod **invokeAll()** koristi promenljiv broj argumenata tipa ForkJoinTask