1. Šta je rekurzija?

Rekurzija je metoda koja poziva samu sebe. U pojedinim slučajevima rekurzija znatno olakšava rešavanje problema. Svaki problem koji se može rešiti rekurzijom, može se rešiti i petljom.

1. Šta podrazumeva rekurzivni način pristupa rešavanju problema?

Ovaj način podrazumeva da imamo metodu, koja poziva samu sebe. Ove metode imaju sledeće karakteristike:

* 1. Koristi se if-else ili switch iskaz koji vodi do različitih slučajeva.
  2. Jedan ili više osnovnih slučajeva koji zaustavljaju rekurziju.
  3. Svaki rekurzivni poziv smanjuje početni problem, dovodeći ga bliže osnovnom slučaju dok ne postane takav slučaj.

Da bi se neki problem rešio rekurzijom, on se mora razdvojiti na podprobleme dok ne dođemo do osnovnog slučaja.

1. Navedite prednosti i nedostatke primene rekurzije?

Prednost rekurzije je što u nekim slučajevima dobijamo znatno kraće i preglednije rešenje u odnosu na iterativni način rešavanja problema. Nedostatak rekurzije je što zahteva dosta računarskih resursa, jer svaki put kad program poziva metodu, on mora obezbediti prostor za sve lokalne promenjive i parametre.

1. Kada koristite rekurziju, a kada petlju?

To zavisi od prirode problema, i vašeg razumevanja problema. Po pravilu, biramo pristup koji nam se odmah intuitivno nameće, a koje odražava problem. Po pravilu, iterativne metode su efikasnije, i ako je njihova primena jasna u odrešenom problemu, treba ih koristiti.

1. Šta je repna rekurzija?

Pod repnom rekurzijom (tail recursion) podrazumeva se rekurzija koja nema zaostale operacije na čekanju, a koje bi trebalo izvršiti po završetku jednog rekurzivnog poziva. Zbog toga, nema potrebe da se memorišu u steku neki međupozivi. Primenom pomoćnih parametara, nerepni rekurzivni metod se može pretvoriti u repni rekurzivni metod. Ovi parametri sadrže rezultate. Ideja je da se na ovaj način izbegnu operacije na čekanju. Kompajleri mogu da optimiziraju repnu rekurziju.

1. Šta je i kako se koristi pomoćna rekurzivna metoda?

Nerepni metod rekurzije može da se konvertuje u repnu rekurziju upotrebom pomoćnih parametara. Ovi parametri sadrže rezultat. Ideja je da se ubace odložene operacije u pomoćne parametre rako da rekurzivni pozivi više nemaju odloženu operaciju. Možete definisati novi metod pomoćne rekurzije sa pomoćnim parametrima. Na ovaj način se redefiniše originalni metod sa istim imenom ali sa različitim potpisom.

1. Šta je uslov za zaustavljanje rekurzije? Da li je obavezno njegovo korišćenje?

Uslov za zaustavljanje rekurzije je onaj osnovni slučaj do kojeg dolazimo rekurzijom. On je obavezan u svakoj rekurzivnoj metodi kako bi mogla da dođe do kraja izvršavanja, inače će metoda pozivati samu sebe u nedogled.

1. Šta su generičke klase?

Generičke (opšte) klase i metode omogućuju parametrizovanje tipova. To znači da možete definisati neku klasu ili metodu sa generičkim tipom koje kompajler može da zameni konkretnim tipovima. Na primer, Java definiše generičku klasu ArrayList za sortiranje elemenata generičkog (opšteg) tipa. Sa ovom generičkom klasom, možete da kreirate ArrayList objekat koji sadrži stringove i ArrayList objekat koji sadrži brojeve. U ovom slučaju, stringovi i brojevi su konkretni tipovi koji zamenjuju generičke (opšte) tipove.

1. Navedite prednosti primene generičkih klasa i metoda?

Glavna korist od generičkih klasa i metoda je u pronalaženju grešaka u vreme kompilacije umesto u fazi izvršenja programa. Generička klasa ili metod dozvoljava vam da specificirate dozvoljene tipove objekata sa kojima klasa ili metod može da radi. Ako pokušate da upotrebite neki nekompatibilan objekat (tipa koji nije specificiran), kompajler će otkriti grešku.

1. Kako se definišu generičke klase i interfejsi?

Generički tip se definiše za klasu ili interfejs. Kada se klasa koristi pri kreiranju objekta, mora da se koristi i konkretan tip. Generička klasa se definiše na sledeći način:

**public class Generic<E> {**

**private E object;**

**public void set(E o){**

**this.object = o;**

**}**

**public E get(){**

**return this.object;**

**}**

**}**

E - može biti bilo koji objekat.

1. Šta je generička metoda i kako se definiše?

Da bi deklarisali generički metod, treba da postavite generički tip <E> odmah posle ključne reči za vidljivost, odnosno nakon ključne reči static u zaglavlju metode.

**public static <E> void print(E[] list);**

1. Šta je sirovi tip? Šta podrazumeva pojam kompatibilnost unazad?

Sirovi tip je generička klasa koja se koristi bez parametra tipa, naziva se sirovim tipom. Upotreba sirovih tipova omogućava kompatibilnost unazad, tj. kompatibilnost sa prethodnim verzijama Jave

1. Objasnite primenu generičkih džoker tipova. Koje generičke džoker tipove poznajete?

Ukoliko imamo neki Stack<Integer> I pokušamo da ga prosledimo kao parametar metodi koja prima Stack<Number> dobićemo grešku u kompilaciji, i ako je Integer podtip od Number. Kako bi prevazišli ovaj problem moramo koristiti džokere. U ovom slučaju bi koristili ograničeni džoker:

**Stack<? extends Number>**

Postoje tri vrste džokera:

* 1. ? - neograničeni džoker - isto kao ? extends Object
  2. ? extends T - ograničeni džoker
  3. ? super T - džoker sa donjom granicom

1. Šta znači brisanje generičkih tipova?

Kompajler upotrebljava informaciju o generičkom tipu da bi kompajlirao kod, ali ga posle briše. Zbog toga, informacija o genericima nije raspoloživa u vreme izvršenja koda. Ovo omogućava generičkom kodu da ostvari kompatibilnost unazad sa starim kodom koji upotrebljava sirove tipove.

Kada kompajler utvrdi da je neki generički tip upotrebljen bezbedno, on ga konvertuje u sirovi tip.

1. Navedite ograničenja primene generičkih tipova podataka.
   1. Ne može da se kreira instanca (objekat) upotrebom parametra generičkog tipa
   2. Ne možete kreirati niz (array) upotrebom parametra generičkog tipa
   3. Parametar generičkog tipa neke klase nije dozvoljen u statičkom kontekstu
   4. Klase izuzetka ne mogu da budu generičke, tj. ne mogu se kreirati, hvatati (catch) ili izbacivati (throw) objekti sa parametrizovanim tipovima
   5. Ne možete primeniti generičke tipove sa primitivnim tipovima
   6. Ne možete upotrebiti konverziju tipova ili koristiti objekte sa parametrizovanim tipovima
   7. Ne možete preopteretiti (overload) metod sa formalnim tipovima parametara svakog brisanja preopterećenja u isti sirovi tip.
2. Zašto koristimo generičke tipove podataka kod kolekcija: lista, redova, setova…?

Koristimo generičke tipove kako bi mogli u kolekciju da stavimo bilo koj objekat koji nam je potreban u programu.

1. Šta predstavlja Java Collection Framework?

Java Collection Framework je kolekcija stuktura podataka u Javi. Ovu kolekciju čine Liste, Vektori, Stekovi, Redovi čekanja i Prioritetni redovi čekanja.

Kolekcija podataka je zbirka podataka organizovana na neki način. Da bi se definisala neka kolekcija, neophodno je da se definiše odgovarajuća klasa. Ta klasa koristi polja podataka (data fields) za skladištenje podataka i obezbeđuje metode za upravljanje podacima, kao što su ubacivanje, brisanje i pretraživanje.

Java Collections Framework podržava dva tipa kontejnera podataka:

* 1. Kolekcija - za čuvanje kolekcije podataka
  2. Mapa - za čuvanje parova ključ/vrednost

Vrste kolekcija:

1. Skupovi (eng. sets) – za skladištenje grupe elemenata bez ponavljanja (istih elemenata)
2. Liste (eng. lists) – za skladištenje uređene kolekcije elemenata
3. Redovi (eng. queues) – za skladištenje objekata koji se uređuju po principu FIFO, tj. "first-in, first-out", tj. po redosledu ubacivanja i uzimanja elemenata iz kolekcije.
4. Šta je iterator i zašto ga koristimo?

Iterator je klasičan šablon projektovanja (eng. design pattern) za pretraživanje neke strukture podataka bez potrebe da se ulazi u detalje kako su podaci uskladišteni u strukturi podataka.

Iterator obezbeđuje uniformni način za pristupanje elementima različitog tipa u nekoj kolekciji. Metod iterator() u interfejsu Collection vraća primerak interfejsa Iterator. On obezbeđuje sekvencijalni pristup elementima kolekcije primenom metoda next(). Možete koristiti takođe i hasNext() metod da bi proverili da li ima još elemenata u iteratoru, a metod remove() da bi uklonili poslednji element koji je vratio iterator.

1. Šta je List? Koje liste poznajete i po čemu se razlikuju?

Interfejs List nasleđuje interfejs Collection i definiše uređenu kolekciju u kojoj elementi mogu da se ponavljaju u listi.

Klase koje implementiraju ovaj interfejs su ArrayList i LinkedList. ArrayList smešta elemente u niz koji se dinamički kreira. Ako se prekorači kapacitet niza, kreira se novi veći niz u koji se svi elementi manjeg niza prekopiraju. LinkedList smešta elemente u vidu povezane liste.

Ako treba da podržavate slučajni pristup elementima preko indeksa bez ubacivanja ili uklananja elemenata na početku liste, ArrayList je efikasnije rešenje.

Ako vaša aplikacija zahteva ubacivanje ili brisanje elemenata na početku liste, onda bi trebalo da koristite klasu LinkedList.

1. Šta je omogućeno implementacijom interfejsa Comparator i Comparable? Koje tipove podataka implementiraju metode ovih interfejsa?

Comparable definiše metod compareTo() metod, koji služi za upoređivanje dva elementa klasa koje implementiraj uinterfejs Comparable. On omogućava upoređenje objekata iste klase.

Interfejs Comparator obezbeđuje poređenje objekata različitih klasa. Možete definisati neki način upoređivanja, tj. upoređivač, da bi poredili elemente različitih klasa. Da bi ovo uradili, neophodno je da se definiše klasa koja implementira interfejs Comparator<T>. Interfejs ima dve metode, compare i equals:

**public int compare(T element1, T element2);**

Metod **compare()** vraća negativnu vrednost ako element1 je manji od element2, a pozitivnu vrednost ako je element1 veći od element2 i vraća nulu ako su jednaki.

Metod **equal()** vraća true ako su objekti jednaki.

**public boolean equals(Object element);**

1. Da li je moguće koristiti korisne statičke metode za liste i kolekcije? Objasnite, ukoliko je moguće, koja klasa je zadužena za poziv navedenih metoda?

Klasa Collections sadrži statičke metode radi izvršenja zajedničkih operacija i u kolekciji i u listi. Često imate potrebu da sortirate neku listu. Java Collections Framework obezbeđuje statičke metode u klasi Collections koji se koriste za sortiranje liste. Klasa Collections takođe sadrži metode binarySearch, reverse, shuffle, copy i fill za liste, i metode max, min, disjoint i frequency za kolekcije.

1. Šta je Vektor, a šta Stack?

Java Collections Framework je uveden u sistemu Java 2. Pre ovoga, Java je podržavala nekoliko struktura podataka, kao što su i klase Vector i Stack. One su redizajnirane, ali su njihovi metodi zadržani radi kompatibilnosti sa starim programima.

Klasa Vector je je ista kao i klasa ArrayList, sem što poseduje i metode za sinhronizaciju pristupa i promenu vektora., tj. objekata klase Vector. Ovi metodi sprečavaju oštećenje podataka kada se pristupa vektoru i kada se menja dejstvo dva ili više simultana toka. Za aplikacije koje ne zahtevaju sinhronizaciju, upotreba klase ArrayList je efikasnija od upotrebe klase Vector.

Klasa Stack se primenjuje kao potklasa klase Vector. Metod **empty()** je isti kao metod **isEmpty().** Metod **peak()** pristupa elementu na vrhu steka, bez njegovog uklanjanja. Metod pop() uklanja element na vrhu steka i vraća taj element. Metod **push(Object element)** dodaje specificiran element na stek. Metod **search(Object element)** proverava da li je specificiran element u steku.

1. Šta je red? Koje vrste redova poznajete? Kako se kreiraju i šta im je namena?

Red (eng. queue) je struktura podataka koja primenjuje princip ”prvi ušao, prvi izašao”, FIFO (eng. first in, first out). Elementi se dodaju na kraj reda, a uklanjaju se sa početka reda. Kod prioritetnog reda (eng. priority queue) elementima se dodeljuje prioritet. Kada se pristupa elementima, prvo se uklanja iz reda element koji ima najveći prioritet.

Klasa PriorityQueue predstavlja prioritetni red. Po pravilu, prioritetni redovi imaju elemente poređane u skladu sa njihovim prirodnim rasporedom, upotrebom interfejsa Comparable. Element sa najmanjom vrednošću ima najveći prioritet, te se i prvi uklanja iz reda. Ako nekoliko elemenata ima isti prioritet, oni se uklanjaju po slučajnom redosledu. Možete da definišete redosled upotrebom objekta tipa Comparator u konstruktoru **new PriorityQueue(initialCapacity, comparator).**

Metod **offer()** se upotrebljava radi dodavanja elementa u red. Metod je sličan metodu **add()** interfejsa Collection, ali je metod **offer()** bolji za redove. Metodi **poll()** i **remove()** su slični, sem što **poll()** vraća null ako je red prazan, a metod **remove()** izbacuje izuzetak. Metodi **peak()** i **element()** su slični, sem što **peak()** vraća null, ako je red prazan, dok metod **element()** izbacuje izuzetak.

Klasa LinkedList implementira interfejs Deque, koji nasleđuje interfejs Queue. Prema tome, vi možete koristiti i klasu LinkedList da kreirate red. LinkedList je idealna klasa za operacije sa redovima jer je efikasna kod ubacivanja i uklanjanja elemenata sa oba kraja liste.

Interfejs Deque nasleđuje interfejs Queue sa dodatnim metodima za ubacivanje i uklanjanje elemenata na oba kraja reda. Metodi **addFirst(e), removeFirst(), addLast(e), removeLast(), getFirst() i getLast()** su definisani u interfejsu Deque.

1. Napravite razliku između struktura red i stek.

Red funkcioniše po principu first in - frist out, tj. onaj element koji je prvi stavljen u red, prvi izlazi iz njega. Kod steka je princip drugačiji. Prvi izlazi onaj koji je poslednji ubačen.

1. Po čemu se razlikuje prioritetni red od "običnog"?

Prioritetni red za razliku od običnog ređa svoje elemente po prioritetu, tj njihovoj vrednosti. Tako najveći prioritet imaju oni elementi sa najmanjom vrednošću. Ukoliko više elemenata imaju isti prioritet, onda izlaze po slučajnom rasporedu.