DODATNO POGLEDATI - 79, 80, 81, 87, 226, 232, prezentacija-ugnjezdeni(8, 9)

- ako se stek moze dinamicki prosirivati a nema dovoljno mmeorije za to ili nema dovoljno memorije pri kreiranju niti – OUTOFMEMORYERROR
- ako neko izracunavanje zahtjeva veci heap nego sto je to moguce – OUTOFMEMORYERROR
- class loader povezuje prevedene datoteke
- identifikatori se sastoje od slova, brojeva, donje crte, broj ne moze biti na pocetku
- sve kljucne rijeci se pisu malim slovima I njihovo nepravilno koristenje rezultuje greskama pri kompajliranju
- literali u oktalnom obliku(0) I u heksadecimalnom obliku(0X) kao prefix, binarni literali ne postoje
- karakter literal pod jednostrukim navodnicima
- primitivni tipovi se cuvaju na steku
- svaki primitivni tip ima odgovarajucu okruzujucu klasu(wrapper)
- reference na steku, dok su sami objekti na heap-u
- dvije vrste inicijalizacionih blokova: staticki I nestaticki
- staticki inicijalizacioni blok se izvrsava JEDANPUT, prilikom ucitavanje klase. Ovaj blok je oznacen kljucnom rijeci static prije separatora {, a u njemu se mogu inicijalizovati samo staticki atributi klasa
- nestaticki inicijalizacioni blok se izvrsava za svaki objekat koji se kreira I inicijalizuje nestaticke attribute klase(moguce je inicijalizovati I staticke attribute ali se ne preporucuje)
- lokalne promjenjive se NE inicijalizuju podrazumjevanom vrijednoscu prilikom prilikom kreiranje kod poziva meotde tj. kada zapocne izvrsavanje metode, prema tome kompajler ce prijaviti kao gresku svaki slucaj koristenja neinicijalizovane promjenjive
- za lokalne reference vazi isto pravilo kao I za pokalne promjenjive(ako je inicijalizujemo sa null moze se desiti izuzetak NULLPOINTEREXCEPTION, tako da je najbolje da svaka referenca pokazuje na konkretan objekat)
- promjenjive instance su dostupne sve dok se objekat kojem pripadaju koristi, Staticke promjenjive su dostupne sve dok je klasa kojoj pripadaju dostupna, Lokalne promjenjive se kreiraju pri svakom izvrsavanju metode, konstruktora ili bloka I nakon okoncavanju vise nisu dostupne
- pri pohranjivanju veceg u manji primitivni tip kompajler javlja gresku
- byte -> short -> int -> long -> float ->double
- ako je neophodno izvrsiti konverziju sireg tipa u uzi tip to se mora eksplicitno uraditi (int I = (int)5.5) I desava se gubljenje informacija
- u slucaju da pokusaj kastovanja nije legalan, desava se greska pri kompajliranju
- konverzija prosirivanja reference nikad nece baciti izuzetak, dok kod konverzije suzavanja reference moze doci do bacanja CLASSCASTEXCEPTON izuzetka

- Boxing je konverzija primitivnog tipa u okruzujuci tip, a unboxing je konverzija iz okruzujuceg tipa u primitivni tip
- pokusaj boxing konverzije u neodgovarajuci okruzujuci tip rezultuje greskom pri kompajliranju, a unboxing reference okruzujuceg tipa koja ima null vrijednost rezultuje NULLPOINTEREXCEPTION
- rezultat aritmetickog izraza ce stalno biti veci tip, postoji jedan izuzetak kada se sabiraju byte I short vrijednosti rezultat je tipa int jer nikada rezultat ne moze biti tipa byte ili short
- naziv metode I lista parametara metoda cine potpis metode, pri cemu je redoslijed parametara u listi bitan
- bitno je napomenuti da se nazivi klasa I nazivi metoda nalaze u razlicitim prostorima imena. Iz tog razloga, ne postoji konflikt naziva izmedju naziva klasa I naziva meotda, prema tome, moguce je deklarisati metodu koja ima naziv identican nazivu konstruktora(I nazivu klase)
- podrazumjevani konstruktor postavlja vrijednosti atributa na podrazumjevane vrijednosti
- ako u klasi postoji samo konstruktor koji nije podrazumjevani, onda je obazveza programera da implementira I podrazumjevani konstruktor, ako je on I potreban. u slucaju da to nije uradjeno svaki pokusaj koristenja podrazumjevanog konstruktora dovodi do greske pri kompajliranju
- greska pri kompajliranju ako se poziva neka metoda koja ne postoji u klasi ili s epristupa privatnim atributima
- statickim clanovima se pristupa preko imena klase
- reference na nizove se cuvaju na steku, a elementi se cuvaju na heap-u
- int[] niz ili int niz[]
- kreiranjem niza alociran je odgovarajuci memorijski proctor I inicijalizovan podrazumjevanim vrijednostima
- deklaracija paketa(package …) onda ide import, I na kraju deklaracija tipova(klase, interfejsi, enumeracija)
- uvozom nekog paketa se ne uvoze I njegovi podpaketi
- kada se uvezu dva tipa sa istim imenom jedino rjesenje je koristenje njihovog punog kvalifikovanog imena
- { kalkulator k = new Kalkulator(); } nakon izlaska iz datog bloka referenca k vise nece biti dostupna, ali objekat klase Kalkulator ce jos zauzimati proctor na heap-u sve dok se ne ukloni od strane garbage collector-a
- ako ne navedemo ni jedan modifikator pristupa podrazumjeva se package, I ta klasa ili atribut je vidljiva u okviru svog paketa
- nije moguce instancirati objekat apstraktne klase ali je moguce imati reference na apstraktnu klasu pri cemu ta referenca refencira objekat klase nasljednice
- enum tipovi ne mogu biti apstraktni
- enum tipovi su implicitno final I ne mogu biti definisani kao final pomocu kljucne rijeci final

- modifikator final u kontekstu klasa znaci da klasa ne moze biti nasljedjena, a u kontekstu atributa final znaci da vrijednost tog atributa ne moze biti promjenjena nakon inicijalizacije
- promjenjive koje su definisane kao static I final najcesce se koriste za definisanje konstanti
- final promjenjive ne moraju biti inicijalizovane pri deklaraciji ali moraju biti inicijalizovane prije nego sto budu koristene
- abstraktne metode mogu samo biti metode instance I to povlaci da I ta klasa mora biti abstraktna, a staticke metode ne mogu biti abstract
- ako metoda zavrsava bacanjem izuzetka postanje return naredbe nije neophodno
- svi izuzeci osim RUNTIMEEXCEPTION, ERROR I njihovih klasa nasljednica nazivaju se provjereni izuzeci. provjereni izuzetak se mora obraditi
- neprovjereni izuzeci su ERROR I RUNTIMEEXCEPTION I sve njihove klase nasljednice I oni ne zahtjevaju provjeru, kada se desi ERROR izuzetak, oporavak uglavnom nije moguc, a RUNTIMEEXCEPTION se desava kao rezultatprogramerksih gresaka
- CLASSNOTFOUNDEXCEPTION bice bacen kada aplikacija pokusa da ucita klasu na bazi datog naziva, CLONENOTSUPPORTEDEXCEPTION bice bacen kada se klonira objekat a ta klasa ne implementira interface Cloneable, INSTANTIONEXCEPTION bice bacen pri pokusaju nepravilnog kreiranja objekta(npr. instanciranje abstraktne klase ili intefejsa), INTERRUPTEDEXCEPTION bice bacen kada jedna nit prekina drugu nit, NOSUCHFIELDEXCEPTION bice bacen kada neka klasa ne posjeduje zahtjevano polje, a NOSUCHMETHODEXCEPTION bice bacen kada neka klasa ne posjeduje zahtjijevanu metodu, PARSEEXCEPTION bice bacen kada se desi greska pri parsiranju
- finally blok se mora stalno izvrsiti
- ako imamo vise catch blokova I ako je prvi blok na koji ce izuzetak naici u hijerarhiji izndad tipa argumenta nekog od narednih catch blokova tada ce se desiti greska pri kompajliranju jer izuzetak nikada nece moci da doce u nize catch blokove
- vrijednost vracena return naredbom u finally bloku je uvijek primarna u odnosu na vrijednost vracenu return naredbom u try bloku
- ako neka metoda samo prosljedjuje izuzetak dalje na tom put ce biti izvrseni svi finally blokovi
- ako se ne nadje ni jedan catch blok koji moze da obradi baceni izuzetak program staje sa izvrsavanjem
- kada neka klasa naslijedi neku drugu klasu I redefinise neku metodu, ona moze da: baci manje izuzetaka ili izostavi throws klauzulu, baci iste izuzetke, baci izuzetke koji su podklase onih koji su baceni u metodi koja je redefinisana
- kalkulator k = new ProsireniKalkulator(1,2); ako se nad k pozove neka metoda iz prosirenog kalkulatora to ce dovesti do greske pri kompajliranju jer kompajler nema saznanja u vrijeme kompajliranja
- ako hocemo da redefinisemo neku metodu u podklasi onda ta metoda mora da ima isti potpis, povratni tip moze da bude podtip povratnog tipa redefinisane metode, nova metoda ne moze smanjiti dostupnost metode, ali je moze povecati, nova metoda moze baciti sve, nijedan ili podskup provjerenih izuzetaka iz redefinisane metode

- ako redefinisemo neku metodu I bacimo neki Exception koji nije definisan u osnovnoj klasi desice se greska pri kompajliranju
- metoda instance u klasi nasljednici ne moze redefinisati staticku metodu osnovne klase, greska pri kompajliranju
- staticka metoda u klasi nasljednici moze sakriti staticku metodu osnovne klase
- metode gdje se nalazi modifikator final ne mogu biti redefinisane, dovodi do greske pri kompajliranju
- ako metoda u nasljednici ima isti potpis kao I u osnovnoj klasi, to znaci da je redefinisana
- ako metoda u nasljednici ima isti naziv ali razlicitu listu parametara, ona se razlikuje po tipu, redoslijedu ili broju parametara onda to znaci da je ta metoda preklopljena
- iako metode instance u klasi nasljednici ne mogu redefinisati staticku metodu, kada su u pitanju polja, polje instance u klasi moze maskirati staticko polje osnovne klase
- this je samo dostupna u nestatickom kontekstu
- ako koristimo poziv konstruktora koristenjem konstrukcije this() ili super() to mora biti prva naredba u tijelu konstruktora
- this() I super() se ne mogu naci u istoj klasi
- ako ne postoji super() konstruktor to podrazumjevano ubaci kao prvu liniju u konstruktoru, ako u osnovnoj klasi nije definisan nijedan konstruktor nece biti problema jer ce biti podrazumjeavno deinisan podrazumjevani konstruktor, ali ako ima neki konstruktor sa parametrima a nema podrazumjevani dolazi do greske pri kompajliranju
- metode interfejsa moraju imati public dostupnost I klasa ne smije smanjiti njihovu dostupnost
- metode interfejsa moraju stalno da budu implementirane kao metode instance, a ne kao staticke metode
- klasa posjeduje samo jednu implementaciju metode koja moze biti deklarisana u vise interfejsa
- kod interfejsa postoji visestruko nasljedjivanje
- u interfejsima se mogu deklarisati konstante koje su implicitno public, static I final, I te konstante moraju biti inicijalizovane
- operator instanceof vraca false ako je s lijeve strane null, a ako ne postoji veza izmedju ova dva tipa desava se greska pri kompajliranju
- polimorfizam se moze ostvariti nasljedjivanjem i implementacijom
- metoda clone – protected Object clone() throws CloneNotSupportedException ovu metodu moze da redefinise svaka klasa I dobra praksa je da se stavlja na public….ako se clone metoda oslanja na clone metodu Object ta klasa mora da impelementira interface Cloneable ili ce baciti CLONENOTSUPPORTEDEXCEPTION
- meotda equals – public Boolean equals(Object obj) vraca istinitu vrijednost ako I samo ako dvije reference koje se porede referenciraju isti objekat
- metoda hashCode – public int hashCode() ova metoda vraca memorijsku adresu objekta kao podrazumjevanu hash vrijednost objekta, ako se redefinise equals metoda, potrebno je redefinisati I hashCode metodu da bi equals raditi kako treba

- metoda toString – public String toString() vraca tekstualnu reperezentaciju objekta, ako se ne redefinise on ace vratiti NazivKlase@hashVrijednostObjekta
- metoda getClass – public final Class<?> getClass() vraca klasu objekta koja je predstavljena objektom klase java.lang.Class
- metoda finalize – protected void finalize() throws Throwable poziva se nad objektom od strane garbage collector-a kada on otkrije da ne postoji niti jedna referenca na taj objekat
- sve okruzujuce klase imaju dva konstruktora, jedan za primitivne tipove, a drugi za String I taj sa Stringom moze baciti NUMBERFORMATEXCEPTION, isti slucaj je I sa valueOf() metodom koja je staticka u okruzujucim klasama
- okruzujuce klase Integer I Long imaju staticke metode za konverziju u odgovarajucu String reprezentaciju (binarna, oktalna, heksadecimalna) toBinary/Octal/HexString(int/long)
- String objekti su nepromjenjivi, to znaci da se objekat klase String ne moze promijeniti. Sve metode klase String vracaju novi objekat, dok originalni objekat uvijek ostaje netaknut
- kompajler optimizuje rukovanje string literalima. Svi string literali sa identicnom sekvencom karaktera dijele samo jedan objekat klase String
- objekti kreirani pomocu konstruktora klase String su uvijek novi objekti, nezavisno od toga koju sekvencu karaktera sadrze
- klase StringBuilder I StringBuffer implementiraju promjnejivu sekvencu karaktera, pored mogucnosti promjene karaktera moze se promijeniti I kapacitet
- StringBuilder I StringBuffer nisu u nikakvoj vezi sa klasom String I ne mogu se cak ni eksplicitno kastovati
- String I StringBuffer su sinhronizovane klase, a StringBuilder nije, to je jedina razlika izmedju StringBuffer I StringBuilder klase, StringBuilder je brzu
- StringBuilder I StringBuffer klase ne redefinisu equals() I compareTo() metode I zbog ovoga se ne mogu porediti…jedini nacin da se uporede su da se pretvore u String I onda da se uporede
- StringBuilder I StringBuffer imaju metode insert(), append(), delete() sa svim mogucim kombinacijama
- StringBuilder I StringBuffer redefinisu metodu toString() I tako se najlakse moze konvertovati u String
- staticke promjenjive I tranzijentne promjenjive nece biti upisane u tok pri procesu serijalizacije
- ako neki objekat sadrzi reference na neki drugi objekat, da bi se izvrsila serijalizacija I taj drugi objekat mora biti serijalizabilan ili ce se desiti NOTSERIALIZABLEEXCEPTION
- objekti klasa koje nasljedjuju serijalizabilnu kalsu uvijek su serijalizabilni
- postoje I alternativni nacini za serijalizaciju objekata, npr. da implementiramo dvije metode writeObject() I readObject() – private void writeObject(ObjectOutputStream oos) throws IOException I private void readObject(ObjectInputStream ois) throws Exception

- drugi alternativni nacin je koristenjem Externalizable interface I implementacijom dvije metode writeExternal() I readExternal() – public void writeExternal(ObjectOutput out) throws IOException I public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException, u ovom slucaju je obavezno postojanje podrazumjevanog konstruktora u klasi ciji se objekat serijalizuje, u slucaju da ovaj konstruktor ne postoji desava se INVALIDCLASSEXCEPTION
- kod generika se prevodi nezavisno, postoji samo jedna klasa, bez obzira na broj poziva generika, Stek<Disk> stekDiskova I Stek<String> stekStringova, kada bi napisali stekDiskova.getClass() == stekStringova.getClass() dobija se true, sto nam pokazuje da su ovo sve iste klase
- sirovi tip za klasu Stek<String> je Stek, dozvoljeno je napisati Stek stekStringova = new Stek() I ovo predstvalja sirovi tip, ovo je omoguceno zbog kompatibilnosti sa starom Javom, jer tamo nije bilo generickih tipova…kada se koriste sirovi tipovi objekat Stek vraca objekat tipa Object
- formalni parameter generika se ne moze koristiti kao tip statickog polja genericke klase, u statickim metodama(povartni tip, tip parametra metode, tip lokalne promjenjive), u statickim inicijalizacionim blokovima date klase, a razlog za ovo je zato sto prevodilac prevodi sve ovo u jednu klasu(sirovi tip)
- sttaickim metodama u generickoj klasi se ne moze pristupiti sa Stek<String>.m(), mora ici sa Stek.m();
- u generickoj klasi A<T> nije dozvoljeno kreirati objekat tipa T ili niz takvih objekata
- argumenti generika mogu biti klase, interfejsi I nizovi(cak I nizovi primitivnih tipova), ali ne mogu biti samo primitivni tipovi
- zbog jednostavnijeg pisanja moguce je I napisati Stek<String> stekStringova = new Stek<>() I ovo ce preuzeti tip koji je sa lijeve strane znaka =
- ako je D podtip od B, to ne znaci da je G<D> podtip od G<B>
- genericka klasa ne moze biti podklasa Throwable klase
- I ugnjezdena klasa moze biti genericka
- parameter generika se moze ograniciti sa gornje strane class G<T extends B & I1 & I2>, ovo znaci da T mora da naslijedi B, implementira I1 I I2, a ako se nista ne navede ogranicenje je klasa Object
- u generickim tipovima, promjenjivoj T se ne moze pristupiti iz statickog konteksta, ona se isto ponasa kao bilo koja druga promjenjiva
- kada se zamijene tipske promjenjive sa stvarnim tipovima nastaju razliciti parametrizovani tipovi koji nemaju veye jedan sa drugim tj. ne postoji nikakva veza izmedju njih
- genericki tip koji nije oznacen kao final moze biti naslijedjen
- GenericSub<T> extends GenericBase<T> ovo nam obezbjedjuje da ce tipske promjenjive biti zamijenjene istim stvarnim tipom u obe klase
- GenericBase<Integer> sub2 = new GenericSub<String>() ovo ce dovesti do greske pri kompajliranju jer ne postoji veza izmedju supertip-podtip
- moguce je da genericki tip naslijedi negenericki, takodje je moguce I da konkretan tip naslijedi parametrizovani tip - IntegerHolder extends GenericHolder<Integer>

- JVM ne zna za genericke tipove I kompajler kada prevodi genericku klasu on informacije o tipskim promjenjivim brise I ubacuje kostovanje u konkretne promjenjive
- kompajler ce prijaviti unchecked upozorenje ako koristenje genericke klase bez parametrizacije moze rezultirati potencijalnim probemom za vrijeme izvrsavanja npr. GenericHolder holder = new GenericHolder<Integer>()
- problem je sto G<D> nije podtip tipa G<T>, gdje je D podtip tipa T I zato se uvode dzoker znakovi, kada bi gledali nizove D[] je podtip od T[]
- ako stavimo G<?> kao formalni parametar neke metode, onda se toj metodi moze poslati bilo koji tip, npr. G<String>, G<Integer>, G<Object>
- postoje dva tipa ogranicenja sa gornje I sa donje strane, sa gornje strane se moze ograniciti sa G<? extends E> to znaci da je E najvisi tip u hijerarhiji sa kojim postoji kompatibilnost, sa donje strane se moze ograniciti sa G<? super E> to znaci da je E najnizi tip u hijrarhiji sa kojim postoji kopatibilnost…nije moguce istovremeno dzoker ograniciti sa donjom I gornjom granicom, takodje se ne moze granica sastojati od vise tipova
- mozemo da koristimo I dzoker znak ? koja oznacava bilo koji tip, tako je GenericHolder<?> nastala od genericke klase GenericHolder kojem je kao parameter stvarnog tipa proslijedjen dzoker znak ?
- GenericHolder<? super Integer> oznacava familiju tipova koji nastaju parametrizacijom generickog tipa GenericHOlder<T> gdje je tipska promjenjiva T mijenja stvarnim tipom Integer ili nekom supertipom Integer-a
- GenericHolder<? extends Number> oznacava familiju tipova koji nastaju parametrizacijom generickog tipa GenericHOlder<T> gdje je tipska promjenjiva T mijenja stvarnim tipom Number ili nekim podtipom Number-a
- Dzoker znakovi se ne mogu koristiti u izrazima za kreiranje instanci, npr. GenericBase<? extends Long> gb1 = new GenericBase<?> ce dovesti do greske pri kompajliranju a ovo se desava jer ? nije konkretan tip, ista ova situacija se desava I pri pokusaju instanciranja interfejsa
- ova situacija je dozvoljena - GenericBase<? extends Long> gb1 = new GenericBase<Long>
- List test = new ArrayList<String>() I ovo je raw tip, nema veze sto je inicijalizovan sa desne strane kada je referenca raw tip
- ako je raw tip proce kompajliranje ali ce se desiti CLASSCASTEXCEPTION ako je konkrentni objekat koji se kastuje drugog tipa nego u taj koji se kastuje, a ako je parametrizovan tip onda ce se desiti greska pri kompajliranju ako se pokusa dodati neki drugi tip u datu kolekciju
- ako klasa niti nasljedjuje neku drugu klasu onda je nemoguce da ta klasa naslijedi I klasu Thread tada je potrebno da ta klasa implementira interface Runnable
- BLOCKED – ceka da dobije monitor zeljenog objekta, WAITING – ceka na notify ili join, TIMED_WAITING - nit spava, ceka na notify ili join, WAITING – nit moze u ovom stanju da bude neograniceni vremenski period
- prioritet niti moze se moze definisati MAX_PRIORITY, NORM_PRIORITY, MIN_PRIORITY
- ako nit spava, I ako se desi interrupt desava se INTERRUPTEDEXCEPTION

- nit prelazi u stanje terminacije kada se okonca metoda run ili kada se baci neki izuzetak u metodi run
- nit koja je usla u sinhronizovanu metodu moze pozivati druge sinhronizovane metode jer da to nije omoguceno nit bi mogla da blokira sama sebe
- ako se sinhornizuje staticka metoda, nit dolazi u posjed java.lang.Class objekta povezanog s klasom
- sinhronizacija se ne moze izvrsiti nad null vbrijednosti, u tom slucaju ce biti bacen NULLPOINTEREXCEPTION
- ako postoji synchronized klok u statickoj metodi, taj blok mora biti sinhronizovan nad objektom java.lang.Class
- ugnjezdena klasa moze da prosiruje proizvoljnu klasu, da implementira proizvoljan intefejs, da bude osnova za prosirivanje, da se deklarise kao final I abstract
- ime ugnezdengo tipa dostupno je direktno u okruzujucem tipu, a izvan mu se pristupa navodjenjem punog imena OkruzujuciTip.UgnjezdeniTip
- ugnjezdena klasa ima pristup svim clanovima okruzujuce klase cak I ako su privatni
- ugnjezedene klase mogu biti I private, protedted I public
- okruzujuca klasa takodje ima pristup svim clanovima ugnjezdene klase
- tipovi ugnjezdeni u interfejse su uvijek(podrazumjevano) javni
- klasa koja nasljedjuje ugnjezdenu klasu ne nasljedjuje prava pristupa
- ugnjezdena klasa moze biti staticka kao I svi drugi clanovi klase, postoje staticke ugnjezdene klase I unutrasnje ugnjezdene klase(nestaticke)
- staticka ugnjezdena klasa ne moze da direktno pristupa nestatickim poljima ili metodama obuhvatajuce klase, moze samo preko refernce tog objekta
- klasa ugnjezdena u interfejs je stalno javna I staticka I ugnjezdeni interfejs je uvijek staticki
- unutrasnja klasa ne moze da sadrzi staticka polja osim final static koja se odmah inicijalizuju, a staticka ugnjezdena klasa moze imati polja static
- ako hocemo da u ugznjezdenoj klasi referenciramo neko polje iz okruzujuce mozemo to da uradimo preko punog imena Okruzujuca.this.polje
- lokalne klase se mogu definisati unutar metoda, unutar konstruktora, unutar inicijalizacionih blokova I oni nisu clanovi okruzujuce klase, one su pristupacne samo u tom bloku gdje su definisane
- reference na objekte lokalne klase se mogu prenositi kao argumenti I vracati kao rezultati metoda
- anonimna klasa ne moze da ima extends I implements klauzulu
- ako ima vise istih default metoda iz vise interfejsa, onda klasa mora da redefinise tu metodu I ako ne redefinise desava se greska pri kompajliranju, a ako postoji samo jedna default ne mora se redefinisati
- naziv klasa moze poceti malim slovom
- ILEGALARGUMENTEXCEPTION je RUNTIMEEXCEPTION I ne mora se baciti u metodi
- samo final konstante mogu biti u switch
- ako se u catch baca novi izuzetak ne mora se nigjde hvatati I ne mora biti u throws klauzuli, a prije tog bacanja se izvrsi finally ako postoji

- try with resources je ista stvar kao I bez ikakvog try bloka
- iako se hvata Exception u metodi moze se bacati izuzetak u metodi
- ako jedan interfejs nasljedjuje drugi interfejs I ako se pojavi metoda sa istim potpisom ali razlicitim povratnim tipom desice se greska pri kompajliranju jer je duplira metoda
- paziti na instanceof da li su u ikakvoj vezi, mogu biti dvije odvojene grane jedne nadklase
- kod enuma ako postoji podrazumjevani konstrkutor onda se pri kreiranju tog tipa poziva konstruktor za svaki izraz u enumu, I enum se ne moze instancirati, enum se ponasa kao primitivni tip, ako se ne inicijalizuje u METODI desava se greska pri kompajliranju, ako se ne inicijalizuje u KLASI onda je null
- konstruktor u enumu mora biti private
- iako se nista ne kreira enum se kreira prilikom ucitavanja tog enuma tj. ponasa se kao staticki kontekst
- bilo kakva veza sa Object clone mora implementirati Cloneable
- kada se klonira objekat dobija se novi objekat, ali ako postoji referenca na neki drugi objekat I ako se ona promijeni mijenja se I u clone(String je izuzetak jer je on IMMUTABLE)
- String string = “string”; String s6 = (“test ” + string).intern() kaze da se provjeri to za vrijeme kompajliranja I onda to predstavlja isti objekat kao I String s1 = “test string”
- ako postoje dvije metode u klasi sa istim potpisom a razlicitim povratnim tipom, kompajler javlja gresku
- klasa nasljednica naslijedi I staticke metode osnovne klase
- Spoljasnja.Unutrasnja u = s.new Spoljasnja.Unutrasnja(); ovo je greska pri kompajliranju…Spoljasnja.Unutrasnja u = s.new Unutrasnja(); ovo je pravilno gdje je s referenca na spoljasnju klasu, mora se upotrijebiti objekat spoljasnje klase jer se unutrasnja klasa veze za spoljasnju
- anonimna klasa bukvalno predstavlja klasu koja je izvedena iz neke poznate klase I prosirena sa nekim dodatnim funkcijama, kada se izvrsava operator new anonimne klase poziva se konstruktor nad klase, moze I podrazumjevani, a moze I sa parametrima
- Ako definisete anonimnu unutrasnju klasu u kojoj želite da koristite objekat definisan izvan nje, prevodilac zahtjeva da taj objekat bude finalan. Zbog toga je argument metode odr() finalan. Ukoliko to zaboravite, dobija se greška pri kompajliranju, a ako se on prosljedjuje kroz konstruktor I nista se ne koristi u anonimnoj klasi onda ne mora da bude finalan, on se moze I koristiti u metodi ako je effective final, a to znaci da mu je samo jednom mijenjana vrijednost prije koristenja, a ako je vise puta onda se desi greska pri kompajliranju
- blok za inicijalizaciju instance predtavlja konstruktor anonimne klase
- ako neka klasa nasljedjuje neku unutrasnju klasu ona ne moze da ima podrazumjevani konstruktor I to ce dovesti do greske pri kompajliranju, mora da primi na reference spoljasnje ciju unutrasnju nasljedjuje I u tom konstrkutoru mora biti ReferencaNaSpoljnuKlasu.super();
- kada se implementira metoda iz interfejsa ona mora imati isti povratni tip ili kovarijantni povratni tip, ili ce doci do greske pri kompajliranju jer ta metoda nije

redefinisana, a ako se onda napravi jos jedna metoda sa istim potpisom ali razlicitim povratnim tipom, doce do greske pri kompajliranju jer metoda sa takvim potpisom vec postoji

- toString() metoda throwable klase je ImeKlase : poruka , ili samo Ime klase ako poruka nije podesena
- paziti na kastovanje u privatnu unutrasnju klasu, to nece proci ako main nije u toj okruzujucoj klasi koja sadrzi tu unutrasnju
- kod generickih tipova se ne moze: pozvati konstruktor paramtra, ne mogu nizovi, ne moze instanceof zato sto je to sve njemu isto
- jedini put kada se javalja greska pri kompajliranju kod raw tipova je kod kod unboxinga, npr imamo raw listu I ubacimo 5 kao broj I onda hocemo da izvrsimo int a = lista.get(0) I desava se unboxing I Object je I greska pri kompajliranju
- imamo primjer List<? extends Bird> lista = new ArrayList<Bird>(); u ovakvu kolekciju ne moze se nista dodavati NIGDJE
- imamo primjer List<? super Bird> lista = new ArrayList<Bird>(); u ovakvu kolekciju moze se dodavati jer u ovom slucaju znamo cijelu strukturu I nece se desiti CLASSCASTEXCEPTION
- List<X super B> ovo ne moze jer super samo moze ici sa dzoker znakom
- public class G<?> pri definiciji klase ovo ne moze
- metoda instance se ne moze redefinisati statickom metodom u klasi nasljednici
- u intefejsu staticke I defulat metode moraju biti definisane ili se desava greska pri kompajliranju
- staticka unutrasnja klasa moze se instancirati bez  objekta okruzujuce klase
- instanceof interfejsa stalno radi, nikada se nece desiti greska pri kompajliranju
- Outer.Inner I = new Outer().new Inner(); ovo prolazi
- poslije definicije enum vrijednosti ako postoje dodatni parametri I metode mora postojati ;
- ako je u pitanju lokalna unutrasnja klasa(klasa unutar metode) ili anonimna klasa onda ona NE MOZE DA MIJENJA SAMO LOKALNE PROMJENJIVE, POLJA MOZE
- Byte – 1B ; Short – 2B ; Int – 4B ; Long – 8B ; Float – 4B ; Double – 8B ; Char – 2B ; Boolean – 1bit