Java – Teoretske cake

1. Prvi karakter u identifikatoru(nazivi klasa, promijenjivih, metoda) ne smije biti broj
2. Ne moze se konvertovati siri tip na uzi

```
long a = 5L;

int b = a;
```

3. Prethodno se moze izvrsiti kastovanjem

```
long a = 5L;

int b = (int)a;
```

4. Moguce je vrsiti konverziju prema gore, odnosno iz podtipa u supertip sto se vrsi implicitno, a kod konverzije prema dole (suzavanje), iz supertipa u podtip potrebno je vrsiti kastovanje
5. **Lokalne promijenjive (koje nisu parametri klase, vec se nalaze kao pomocne promijenjive u metodama, konstruktorima I blokovima) se ne inicijalizuju implicitno prilikom kreiranje, tako da ih je potrebno rucno (eksplicitno) inicijalizovati inace kompajler prijavljuje gresku. Isto vazi za lokalne reference samo sto je njih moguce inicijalizovati null referencom,ali onda moze doci do greske prilikom izvrsavanja**
6. Kod operatora “=”, ako su operandi reference, kopira se samo sadrzaj reference a ne kompletni objekti na koje ukazuju
7. Ako je jedan od operanada kod operatora “=” klase String, cio izraz je String
8. Break – prekida tijelo tekuce ciklicne structure I izlazi iz nje, a continue prekida tijelo tekuce ciklicne structure I otpocinje sledecu iteraciju petlje
9. Moguce je deklarisati metodu koja ima naziv identican nazivu konstruktora (I nazivu klase)
10. Moguce je da se u jednoj datoteci nadje vise definicija razlicitih klasa
11. Naziv datoteke treba da bude identican nazivu klase, I sa ekstenzijom .java
12. klasa koja ima public vidljivost mora biti definisana u datoteci sa nazivom koji je identičan nazivu klase i ekstenzijom .java (ovo pravilo određuje da izvorna

datoteka ne može sadržavati više od jedne public klase, tj. ako se u jednoj izvornoj datoteci nalaze definicije više klasa, onda samo jedna može biti public i izvorna datoteka će imati naziv koji odgovara nazivu public klase (sa ekstenzijom .java)). Svaka definicija klase iz iste izvorne datoteke kompajlira se u zasebnu datoteku sa .class ekstenzijom

13. može postojati samo jedna deklaracija paketa u datoteci sa izvornim kodom, tj. jedna klasa se ne može nalaziti u više paketa
14. korištenje tipova (klasa i interfejsa) moguće je i bez uvoza, ali se u tom slučaju navodi puno kvalifikovano ime tipa
15. deklaracija uvoza u datoteci sa izvornim kodom se mora nalaziti iza deklaracije paketa
16. identifikatori klase mogu maskirati statičke članove koji se uvoze, u ovim slučajevima jedini način pristupa određenoj statičkoj metodi ili statičkom atributu jeste navođenjem njenog/njegovog punog kvalifikovanog imena
17. iz statičkog konteksta može se pristupiti samo statičkim članovima
18. kako se statički kod ne izvršava u kontekstu objekta, tako i reference this i super nisu dostupne iz statičkog konteksta
19. iz nestatičkog konteksta su uvijek dostupni i statički članovi
20. lokalne promjenljive metode obuhvataju formalne parametre metode, kao i promjenljive koje su deklarisane unutar tijela metode I kreiraju se iznova, pri svakom pozivu metode
21. bitno je napomenuti da objekti nemaju isti opseg vidljivosti kao primitivni tipovi i reference, jer objekat će postojati na heap-u sve dok ne bude uklonjen od strane garbage collector-a
22. modifikator pristupa public može se koristiti za deklaraciju top-level tipova (klasa, interfejsa i enumeracija) u paketima, kako bi one mogle biti dostupne kako iz svog paketa, tako i iz drugih

Public – dostupan iz svih paketa Podrazumijevani – dostupan samo iz svog paketa

23. modifikator abstract se koristi u deklaraciji klase kako bi označio da klasa ne može biti instancirana, tj. kako bi označio da je klasa apstraktna

24. svaka klasa može biti deklarisana kao apstraktna klasa, ali klase koje imaju jednu ili više apstraktnih metoda (metode deklarisane ključnom rječju abstract i koje nemaju tijelo) moraju biti deklarisane kao apstraktne klase. Ovakve klase se mogu koristiti kako bi specificirale određeno ponašanje, ali se klasama nasljednicama ostavlja da obezbijede odgovarajuću implementaciju. Klase nasljednice, da bi mogle biti instancirane, moraju obezbijediti implementaciju svih apstraktnih metoda nasljeđenih iz apstraktne klase – u suprotnom, i klase nasljednice moraju biti deklarisane kao apstraktne
25. interfejsi specificiraju samo apstraktne metode, bez implementacije bilo koje metode – interfejsi su po svojoj prirodi implicitno apstraktni i ne mogu biti instancirani
26. interfejs je moguće deklarisati kao apstraktan, korištenjem ključne riječi abstract, ali je to neoptrebno i predstavlja redundansu, dok enum tipovi ne mogu biti deklarisani kao apstraktni
27. modifikator final se koristi u deklaraciji klase kako bi označio da klasa ne može biti nasljeđena, tj. kako bi označio da ne mogu postojati klase nasljednice date klase
28. samo klase koje su u potpunosti definisane, tj. kod kojih su sve metode implementirane, mogu biti deklarisane kao final klase - prema tome, klasa ne može u isto vrijeme biti deklarisana i kao abstract i kao final
29. navođenjem modifikatora pristupa članova, klasa kontroliše koji od njih su dostupni drugim klasama (klijentima), modifikatori pristupa članova su: public, protected, private i nespecificirani (prijateljski, friendly) pristup
30. bitno je napomenuti da modifikatori pristupa članova klase imaju značenje samo ako je klasa ili klasa nasljednica dostupna klijentima (drugim klasama); ako klasa nije deklarisana kao public klasa, onda public modifikator pristupa članova date klase nema nikakvo značenje za klase iz drugih paketa
31. specijalni karakteri [ ] { } ( ) , . * + ? $ ” ’ \ ^
32. u interfejsu iako metoda nema modifikator u klasi naslednici mora biti napisan public inace ce kompajler prijaviti gresku,jer je podrazumijevani frendly sto znaci smanjili smo prava pristupa

33. metoda join blokira izvrsavanje niti odakle je pozvana(u vecini slucaja main koja kreira sve ostale niti) sve tok metoda nad kojom je pozvana ne zavrsi sa radom
34. ako se kod stringova za poredjenje koristi operator = poredi se da li ta dva stringa referenciraju isti objekat,a ako se koristi equals poredi se da li su im vrijednosti objekata iste
35. ako imamo String s=”abc”; -svi stringovi npr s2,s3 kojima je dodjeljen abc referenciraju isti objekat samo ima jedan stvarni objkat “abc”
36. ako klasa implementira interfejs serializable a ima referencu koja referencira klasu koja ne implementira serializable pri serijalizaciji prijavice gresku
37. Klasa koja nasledjuje Externalizable mora biti deklarisana kao public inace se nece moci deserijalizovati prijavice gresku-prvo se izvrsava podrazumijevani konstruktor pa se onda ocitaju vrijednosti-iako smo neke promjenljive deklarisali kao tranzientne pri DEserijalizaciji ce postojati jer se ponovo konstruise objekat pa im normalno mozemo pristupiti
38. Reference na steku,objekti na hipu
39. Nizovi u javi predstavljeni kao objekti I mogu imati primitivne tipove kao I objekte
40. Ime klase mora odgovarati imenu datoteke
41. U okviru istog paketa samo jedna klasa moze biti deklarisana kao public Nije tacno!
42. Java ne podrzava literale u binarnom brojnom sistemu
43. U jednoj datoteci moze biti samo jedna naredba package….tj jedna datoteka ne moze biti u dva paketa
44. Deklaracija uvoza u datoteci mora biti iza deklaracije pakete tj

    package p;

    pmport java.io.*; nikako obratno
45. Nad boolean tipom se moze primjeniti & | ^ (tj svi osim ~ negacije)
46. etfbl.net net.etfbl
47. etf-bl.net net.etf_bl
48. student.etfbl.int int_.etfbl.student
49. paket.24sata.info info._24sata.paket

50. svaka klasa moze biti deklarisana kao apstraktna,ali klasa koja ima bar jednu apstraktnu metodu obavezno mora biti deklarisana kao apstraktna
51. ako klasa naslednica nasledjuje apstraktnu klasu I ako ne implementira sve njene metode I ona mora biti deklarisana kao apstraktna.
52. Interfejsi su apstraktni I ne mogu se instacirati
53. Klasa kod koje je modifikator final njene metode se ne mogu redefinisat prijavljena ce biti greska I ta klasa mora imati implementirane sve metode sto znaci klasa u isto vrijeme ne moze biti I final I abstract
54. Jedan clan klase moze imati samo jedan modifikator pristupa inace greska
55. Enum tipovi ne mogu biti apstraktni
56. Modifikator pristupa apstraktne metode u nekoj klasi ne moze biti private jer se klasa ne moze redefinisati
57. Kako staticke metode ne mogu biti redefinisane tako one ne mogu biti deklarisane kao apstraktne
58. Final metode ne mogu biti nekompletne tj ne implementirane
59. Error I runtime izuzeci ne moraju ispuniti catch ili specity zahtjev jer su necekirani izuzeci
60. Catch blok ne mora postojati,ali mora biti finally
61. Catch ili finally se ne mogu pojaviti prije try bloka I ne mogu se pojaviti bz try bloka
62. Kada redefinisemo metodu mozemo baciti sve,manje,ni jedan ili podklase izuzetaka
63. Ako se u try catch bloku hvata izuzetak roditeljske klase pa klase naslednice prijavice se greska
64. Konstruktori I inicijalni blokovi se ne mogu naslediti
65. U drugim paketima se nasledjuju samo varijable koje imaju public I protected dostupnost
66. Klasa koja nije definisana kao public nece biti dostupna izvan svog paketa
67. REDEFINISANJE METODE
    Metoda mora imat isti naziv,tip,broj I redosled parametara
    Povratni tip moze biti kovarijantan tip
    Nova metoda ne moze smanjiti dostupnost,ali je moze povecati

Moze baciti sve,ni jedan,ili podskup provjerenih izuzetaka koji su specificirani u throws klauzuli metode koju redefinise
SAMO U KLASI NASLJEDNICI

68. Staticni clanovi zive na nivou klasa
69. U klasi naslednici se ne moze redefinisati metoda roditeljske klase bice greska
Nije tacno!
70. Staticka metoda u klasi nasljednici moze sakriti staticku metodu roditeljske klase
71. Apstraktna klasa moze imati staticka polja I metode
72. PREKLAPANJE
Metode imaju isti naziv,ali razlicitu listu parametara
Lista parametara se moze razlikovati kako po tipu tako I po broju
Nije dovoljno da se metode razlikuju samo po povratnom tipu da bi bileprekloplјene
I metode instance I staticke metode mogu biti prekloplјene kako u osnovnoj tako I u klasi nasljednici
U ISTOJ KLASI ILI U KLASI NASLJEDNICI
73. Polje osnovne klase ne moze biti redefinisano u klasi nasljednici,ali moze biti maskirano
74. Polje u klasi nasljednici moze maskirati staticko polje osnovne klase
75. Tip polja nije bitan vazan je samo naziv
76. Staticka metoda klase nasljednice moze maskirati staticku metodu roditeljske klase ako vaze isti uslovi kao I za redefinisanje metoda instanci
77. Ako su im potpisi isti a zahtjevi kao throws klauzule ili tipa nisu isti prijavice se greska
78. Ako su potpisi metoda razliciti onda je rijec o preklapanju,a ne o maskiranju
79. Staticka metoda u klasi nasljednici ne moze maskirati metodu instance osnovne klase
80. Prva linija u polju konstruktora ili this ili super ne moze oboje
81. Kod preklapanja metoda NEMA OGRANICENJA ZA POVRATNI TIP,ZA IZUZETKE I ZA DOSTUPNOST
82. Ako klasa nema konstruktor kompajler ce jos ugraditi podrazumijevani BEZ PARAMETARA

83. Interfejsi mogu biti I bez tijela
84. Sve metode interfejsa su implicitno abstract I public
85. Sve konstante u interfejsu su public static final
86. Metode interfejsa koje implementira klasa moraju imati public dostupnost
    Klasa ne moze specificirati novi izuzetak ako ga nema u metodi interfejsa
87. Kriterijumi za redefinisanje metode vaze I pri implementaciji interfejsa
88. Metode interfejsa uvijek moraju biti implementirane kao metode instance a ne kao staticke metode
89. Metode interfejsa ne mogu biti ni staticke ni filal ni private ni protected
    Podrazumijevano u interfejsu su public abstract
    Podinterfejs moze redefinisati metode svojih superinterfejsa pri tome redefinisane metode nisu nasledjene
90. Deklaracije apstraktnih metoda mogu biti I preklopljene analogno preklapanju metoda kod klasa
91. KONSTANTE U INTERFEJSIMA MORAJU BITI INICIJALIZOVANE.
92. KONSTANTE U KLASAMA MORAJU BITI INICIJALIZOVANE PRIJE KORISTENJA
93. Metoda toString vraca NazivKlase@hes vrijednost objekta
94. Klasa StringBuffer je sinhronizovana
95. Instance klase FILE su nepromjenljive
96. SERIJALIZACIJA
    Serijalizujemo objekte,a ne klase
    Tranzijentne I staticke clanove ne mozemo serijalizovati implicitno
97. KOKNURENTNO PROGRAMIRANJE
98. Metodu sleep staviti u try cacth blok
99. Interferencija niti
100. Greske usljed nekonzistentnosti memorije
101. Happens before relazija
    Sinhronizacija,volatile promjenljive start I join
    Monitor moze biti u posjedu samo jedne niti u jednom vremenskom periodu
102. Sinhronizaciju mozemo izvrsiti preko sinhronizacionih metoda I preko sinhronizacionih blokova
103. Synchronized se koristi samo za metode ne moze za polja
104. Sinhronizacija konstruktora nije moguca

105. REETRANT sinhronizacija
Omogucava nitima da vise puta dodju u posjed monitora
Tamo gdje sinhronizacioni kod poziva metodu koja takodje ima sinhronizacioni kod I oba segmenta korisne isti monitor
Bez ovoga bi se tesko izbjeglo sopstveno blokiranje niti
Postoje tri vrste blokiranje mrtvo,zivo I izgladnjivanje
Objekat je immutable ako njegovo stanje posle kreiranje ne moze biti promjenjeno
Za postizanje immutable:
Ne smiju biti seteri
Polja moraju biti private I final
Deklarisati klasu kao final a moze I privatni konstruktor
GENERICKI TIPOVI
106. Za rjesavanje compile time bug-ova
107. Run time bug-ova
108. Daju odredjenu stabilnost
109. Detekcija odredjenih gresaka
110. Tipskoj promjenljivoj T se ne moze pristupiti iz statickog konteksta
111. Genericki tip koji nije deklarisan kao final moze biti nasljedjen
112. Kod mape I kljucevi I vrijednosti moraju biti objekti
113. modifikator public – označava da je član klase vidljiv za sve klase u bilo kojem paketu programa – ovo važi i za članove instance i za statičke članove
114. modifikator protected – označava da je član klase vidljiv samo za klase u istom paketu i za sve klase nasljednice u bilo kojem paketu – sve ostale klase ne mogu pristupiti protected članovima
115. modifikator private – označava da je član klase vidljiv samo unutar svoje klase – ovo važi i za klase nasljednice, bez obzira da li se one nalaze u istom ili drugom paketu
116. paketski ili prijateljski (friendly) pristup – ako modifikator pristupa člana nije specificiran – podrazumijevani modifikator označava da je član klase vidljiv samo od strane klasa u istom paketu (podrazumijevani modifikator je restriktivniji od protected modifikatora)

117. modifikator static – statički članovi pripadaju klasi u kojoj su deklarisani i nisu dio niti jedne instance klase
118. modifikator final – promjenljiva u čijoj deklaraciji se nalazi modifikator final je konstanta. Njena vrijednost ne može biti promjenjena nakon inicijalizacije, tj. vrijednost promjenljive primitivnog tipa ne može biti promjenjena, kao ni vrijednost reference (final reference uvijek pokazuju na isti objekat). Ključna riječ final ne može uticati na to da stanje objekta na koji ukazuje final referenca ne bude izmijenjeno
119. promjenljive koje su static i final se najčešće koriste za definisanje konstanti
120. promjenljive definisane unutar interfejsa su implicitno final promjenljive
121. final promjenljiva ne mora biti inicijalizovana pri deklaraciji, ali mora biti inicijalizovana prije nego što bude korištena
122. final promjenljive obezbjeđuju da vrijednost ne može biti promjenjena, dok final metode obezbjeđuju da ponašanje ne može biti promjenjeno
123. modifikator pristupa apstraktne metode ne može biti private, jer u tom slučaju ne bi bilo moguće da klasa koja nasljeđuje apstraktnu klasu redefiniše apstraktnu metodu i obezbijedi njenu implementaciju
124. samo metode instance mogu biti deklarisane kao apstraktne –kako statička metoda ne može biti redefinisana, nije dozvoljeno ni njeno deklarisanje kao apstraktne metode
125. final metoda ne može biti apstraktna, jer ne može biti nekompletna
126. metode specificirane u interfejsu su implicitno apstraktne metode, tako da modifikator abstract nije potrebno ni navoditi u deklaraciji metode interfejsa
127. modifikator synchronized može biti korišten pri deklaraciji metoda, ne i promjenljivih
128. native metode su metode čija implementacija nije definisana u Java programskom jeziku, već u nekom drugom programskom jeziku (npr., C, C++, ...)
129. modifikator transient može biti korišten pri deklaraciji promjenljivih, ne i metoda, ove promjenljive se koriste kada je potrebno sačuvati stanje objekta koristeći serijalizaciju
130. modifikator volatile može biti korišten pri deklaraciji promjenljivih, ne i metoda; ovaj modifikator se koristi kada je potrebno upozoriti kompajler da ne

vrši optimizacije nad poljem koje je deklarisano kao volatile, a koje bi mogle dovesti do nepredvidivih rezultata kada se polju pristupa od strane više niti

131. Klasa Throwable je roditeljska klasa svih grešaka (klasa Error i njene klase nasljednice) i svih izuzetaka (klasa Exception i njene klase nasljednice)
132. Samo objekti koji su instance ove klase ili neke od njenih klasa nasljednica mogu biti bačeni od strane JVM ili putem throw naredbe. Slično, samo ova klasa ili neka od njenih klasa nasljednica može biti tip argumenta catch klauzule
133. nakon try bloka, mora doći barem jedan catch blok ili finally blok
134. moguće je da catch blok ne postoji, ali u toj situaciji finally blok mora biti specificiran
135. blokovi catch i finally moraju da se pojave nakon try bloka i ne mogu da se pojave bez try bloka
136. nakon završetka catch bloka, izvršavanje se nastavlja u finally bloku, ako je ovaj blok specificiran, i to bez obzira na to da li je novi izuzetak bačen u catch bloku.
137. situacija u kojoj catch blok ima argument čiji je tip u hijerarhiji iznad tipa argumenta nekog od narednih catch blokova nije dozvoljena – ovakvu situaciju prijaviće kompajler, jer naredni catch blok ne bi nikad mogao da se izvrši.
138. finally blok – uvijek se izvršava, bez obzira na to da li se desio izuzetak u try bloku i da li je izvršen catch blok
139. Nasljeđivanje članova zavisi od njihove dostupnosti. Ako je članu osnovne klase moguće pristupiti navođenjem naziva tog člana, bez bilo kakve dodatne sintakse (poput ključne riječi super), onda se taj član smatra nasljeđenim. Iz tog razloga privatni, redefinisani i sakriveni članovi osnovne klase nisu nasljeđeni.
140. izvedena klasa nasljeđuje sve public i protected članove roditeljske klase, bez obzira u kojem paketu je izvedena klasa. Ako je izvedena klasa u istom paketu kao i roditeljska, nasljeđuje i friendly članove; članovi koji imaju podrazumijevanu dostpunost u osnovnoj klasi ne mogu biti nasljeđeni od strane klasa u drugim paketima
141. Konstruktori i inicijalizacioni blokovi ne mogu biti nasljeđeni, jer nisu članovi klase

142. U Java programskom jeziku nije dozvoljeno višestruko nasljeđivanje, tj. svaka klasa može imati samo jednu roditeljsku klasu –jednostruko nasljeđivanje
143. nasljeđivanje definiše relaciju „je vrsta“ između osnovne i klase nasljednice – ova relacija podrazumijeva da se vrijednost reference objekta klase nasljednice može dodijeliti referenci objekta osnovne klase, jer objekat klase nasljednice može da se pojavi gdje god se očekuje objekat osnovne klase - ova dodjela podrazumijeva proširivanje reference (eng. widening reference conversion)
144. referenca kalkulator1 se može koristiti za poziv metoda nad objektom klase ProsireniKalkulator koje su nasljeđene iz klase Kalkulator

```
Kalkulator kalkulator1 = new ProsireniKalkulator(1,2); // 1
int zbir = kalkulator1.zbir();                         // 2
// int proizvod = kalkulator1.proizvod();              // 3
```

145. bitno je primijetiti da kompajler u vrijeme prevođenja programa ima saznanje o deklarisanom tipu reference
    – na osnovu toga što je tip reference kalkulator1 Kalkulator, kompajler ograničava da samo metode iz klase Kalkulator mogu biti pozivane putem ove reference
    – za vrijeme izvršavanja referenca kalkulator1 referenciraće objekat klase ProsireniKalkulator – iako referenca kalkulator1 referencira objekat klase ProsireniKalkulator nije moguće pozvati metodu proizvod, jer kompajler u vrijeme prevođenja nema saznanje o tome koji objekat će referenca kalkulator1 referencirati, tj. kompajer ima saznanje samo o deklarisanom tipu reference (poziv metode proizvod u ovom slučaju rezultiraće greškom pri kompajliranju)
146. klasa nasljednica može redefinisati metode osnovne klase - redefinisanje metode omogućava da klasa nasljednica obezbijedi vlastitu implementaciju metode koju bi inače naslijedila iz osnovne klase
147. redefinisana metoda osnovne klase se u ovoj situaciji ne nasljeđuje, a nova metoda u klasi nasljednici mora da ispunjava sljedeće:
    – mora da ima isti potpis (naziv metode, tip, broj i redoslijed parametara),

– povratni tip nove metode može biti podtip povratnog tipa redefinisane metode (kovarijantni povratni tip),
– nova metoda ne može smanjiti dostupnost metode, ali je može povećati,
– nova metoda može baciti sve, nijedan ili podskup provjerenih izuzetaka (uključujući i njihove podklase) koji su specificirani u throws klauzuli redefinisane metode

148. Metoda instance u klasi nasljednici ne može redefinisati statičku metodu osnovne klase – pokušaj kompajliranja dovešće do greške - statička metoda u klasi nasljednici može sakriti statičku metodu osnovne klase

149. Metode u čijoj deklaraciji se nalazi modifikator final ne mogu biti redefinisane – pokušaj redefinisanja ovakve metode dovešće do greške pri kompajliranju

150. Apstraktne metode u klasama nasljednicama koje nisu apstraktne moraju biti redefinisane, tj. implementacija ovih metoda mora biti obezbjeđena

151. Modifikator private u deklaraciji metode označava da metoda nije dostupna za klase nasljednice, pa iz tog razloga ova metoda ne može biti redefinisana

152. ako klasa ima samo apstraktne metode – onda treba biti interfejs, a ne apstraktna klasa

153. klase koje su deklarisane kao abstract mogu, ali ne moraju, imati apstraktne metode

154. apstraktna klasa može imati statička polja i statičke metode i moze implementirati interface

155. povratni tip nove metode može biti podtip povratnog tipa redefinisane metode – ovaj povratni tip naziva se kovarijantnim povratnim tipom

156. klasa nasljednica mora koristiti ključnu tiječ super kako bi pozvala redefinisanu metodu roditeljske klase

157. preklopljene metode su metode koje imaju isti naziv, ali različitu listu parametara - lista parametara se može razlikovati po tipu, redoslijedu i/ili broju parametara

158. kako povratni tip nije dio potpisa metode, nije dovoljno da se metode razlikuju samo po povratnom tipu da bi bile preklopljene

159. i metode instance i statičke metode mogu biti preklopljene u osnovnoj ili klasi nasljednici

160. ako klasa posjeduje više konstruktora oni su uvijek preklopljeni, jer svi konstruktori imaju isti naziv (koji odgovara nazivu klase), pa se njihovi potpisi razlikuju samo po listi parametara

	redefinisanje	preklapanje
naziv metoda	mora biti identičan	mora biti identičan
lista parametara	mora biti identična	ne smije biti identična
povratni tip	identičan ili kovarijantan	nema ograničenja
izuzeci	svi, nijedan ili podskup izuzetaka redefinisane metode	nema ograničenja
dostupnost	ne može se smanjiti	nema ograničenja
lokacija	samo u klasi nasljednici	u istoj ili u klasi nasljednici
izvršavanje metode određuje	tip objekta referenciran u vrijeme izvršavanja	deklarisani tip reference

161. polja osnovne klase ne mogu biti redefinisana u klasi nasljednici, ali mogu biti maskirana

162. polja se maskiraju tako što se u klasi nasljednici definiše polje sa istim nazivom kao u osnovnoj klasi – u ovakvom slučaju poljima osnovne klase se ne može pristupiti direktno, pa je jasno da se ona ne nasljeđuju – pristup poljima osnovne klase koja se ne nasljeđuju, uključujući i maskirana polja, može se ostvariti korištenjem ključne riječi super

163. za razliku od redefinisanja metoda, gdje metoda instance ne može redefinisati statičku metodu, kod maskiranja polja ne postoji takvo ograničenje – polje instance u klasi nasljednici može maskirati statičko polje osnovne klase

164. tip maskiranih polja nije bitan, važan je jedino naziv polja

165. statička metoda u klasi nasljednici može maskirati statičku metodu osnovne klase, ako su uslovi identični uslovima za redefinisanje metoda instance ispunjeni

166. ako su potpisi metoda identični, a drugi zahtjevi poput povratnog tipa, throws klauzule i dostupnosti nisu ispunjeni, kompajler će prijaviti grešku

167. ako su potpisi metoda različiti, onda je riječ o preklapanju metoda, a ne o maskiranju

168. maskirana statička metoda uvijek može biti pozvana preko imena osnovne klase u kodu klase nasljednice, kako iz statičkog tako i iz nestatičkog konteksta

– moguće je koristiti ključnu riječ super za poziv maskirane statičke metode, u nestatičkom kontekstu u kodu klase nasljednice

169. statička metoda u klasi nasljednici ne može maskirati metodu instance osnovne klase – pokušaj ovakvog maskiranja dovešće do greške pri kompajliranju

170. ključna riječ super označava referencu koju je moguće koristiti u nestatičkom kontekstu, ali samo u klasi nasljednici, radi pristupa poljima i pozivanju metoda osnovne klase

171. pri pozivu metoda korištenjem ključne riječi super, metoda osnovne klase se poziva bez obzira na stvarni tip objekta i bez obzira na to da li je metoda redefinisana u klasi nasljednici

172. konstruktori ne mogu biti nasljeđeni niti redefinisani, ali mogu biti preklopljeni unutar iste klase

173. ulančavanje: korištenjem this i korištenjem super

174. konstrukcija this() sa odgovarajućom listom parametara rezultira pozivom odgovarajućeg konstruktora

175. obavezno je da poziv konstruktora korištenjem konstrukcije this() bude prva naredba u tijelu konstruktora

176. konstrukcija super() se koristi u konstruktoru klase nasljednice radi poziva odgovarajućeg konstruktora neposredne roditeljske klase – koji konstruktor roditeljske klase će biti izvršen zavisi od liste parametara super() konstrukcije – preporuka je da se konstrukcija super() koristi za inicijalizaciju nasljeđenog stanja• konstrukcija super() mora biti prva naredba u konstruktoru i moguće ju je koristiti samo u konstruktoru

177. kako moraju biti prve naredbe u konstruktoru, jasno je da se konstrukcije this() i super() ne mogu pojaviti u istom konstruktoru

178. ako se u konstruktoru klase nasljednice ne nalazi eksplicitan poziv konstrukcije super(), kompajler će implicitno ubaciti poziv ove konstrukcije (bez argumenata) kako bi pozvao podrazumijevani konstruktor roditeljske klase

179. ako roditeljska klasa ne posjeduje podrazumijevani konstruktor, tj. ako roditeljska klasa posjeduje samo konstruktore sa parametrima, onda ubacivanje poziva konstrukcije super() od strane kompajlera u konstruktore

klasa nasljednica neće imati efekta – u ovakvim situacijama desiće se greška pri kompajliranju – da bi se ova greška izbjegla u konstruktorima klasa nasljednica potrebno je eksplicitno pozvati konstruktor roditeljske klase, korištenjem konstrukcije super() sa odgovarajućim argumentima

180. interfejse implementiraju klase, pa iz tog razloga članovi interfejsa implicitno imaju public dostupnost – iz tog razloga modifikator public može biti izostavljen u deklaraciji članova interfejsa
181. da bi klasa implementirala interfejs, mora da implementira sve njegove metode
182. sve konstante definisane u interfejsu su implicitno public, static i final
183. metode interfejsa moraju imati public dostupnost pri implementaciji u klasi (ili njenim klasama nasljednicama)
184. klasa ne može smanjiti dostupnost metode interfejsa, niti može specificirati novi izuzetak u throws klauzuli metode, jer će to dovesti do narušavanja ugovora interfejsa – što nije legalno
185. klasa može implementirati samo neke od metoda interfejsa, tj. klasa može obezbijediti parcijalnu implementaciju interfejsa – u ovom slučaju klasa mora biti deklarasana kao apstraktna
186. metode interfejsa uvijek moraju biti implementirane kao metode instance, a ne kao statičke metode
187. bez obzira koliko interfejsa klasa implementira, direktno ili indirektno, ona posjeduje samo jednu implementaciju metode koji može biti deklarisana u više interfejsa
188. jedan interfejs može da naslijedi jedan ili više drugih interfejsa, tj. kod interfejsa postoji višestruko nasljeđivanje – u deklaraciji interfejsa koji nasljeđuje jedan ili više drugih interfejsa nazivi interfejsa koji se nasljeđuju se specificiraju u zaglavlju interfejsa, nakon ključne riječi extends, i razdvajaju se znakom „ , “
189. podinterfejs nasljeđuje sve metode svojih superinterfejsa, jer su metode interfejsa javne (public)
190. podinterfejs može redefinisati deklaracije apstraktnih metoda nasljeđene od svojih superinterfejsa – pri tome, redefinisane metode nisu nasljeđene

191. iako interfejsi ne mogu biti instancirani, moguće je deklarisati referencu nekog tipa interfejsa
192. u interfejsu se mogu deklarisati konstante koje su implicitno public, static i final - ove konstante moraju biti inicijalizovane
193. konstantama deklarisanim u interfejsu može pristupiti bilo koji klijent (klasa ili interfejs) putem punog kvalifikovanog imena
194. ako klijent (klasa ili interfejs) implementiraju, odnosno nasljeđuju interfejs, onda je konstanti moguće pristupiti navođenjem njenog jednostavnog imena
195. polimorfizam omogućava da reference referenciraju objekte različitih tipova u različito vrijeme tokom izvršavanja
196. poziv privatne metode instance nije i ne može biti polimorfan, jer se takav poziv može javiti samo unutar klase i odnosi se na tačno određenu metodu, što može biti utvrđeno za vrijeme kompajliranja
197. polimorfizam se može ostvariti kako putem nasljeđivanja, tako i putem implementacije interfejsa
198. unutrasnja klasa ima pristup svim varijablama okruzujuce klase cak I privatnim samo nema pristup poljima u mainu.
199. Ako se dva puta pozove start za istu nit desice se greska/izuzetak
200. Frendly modifikator pristupa je jaci/rigorozniji od protected
201. Integer extends Number tako da nije dozvoljeno Integer I = (Number)o;
202. Short,int,char,byte,enum mogu u switch
203. Hes tabela ne dozvoljava ni kljucevi ni vrijednosti da budu null
204. Kada imamo join metodu moramo baciti izuzetak ili imati try catch blok specificiran
205. U hesh setu je potrebno redefinisati metodu equals
206. Unutrasnja neimenovana klasa moze naslediti tacno jednu klasu ili implementirati tacno jedan interfejs
207. Metoda lokalne unutrasnje klase moze biti apstraktna
208.
209. Klasa StringBuffer ne redefinise equals i hesh f-ju
210. Apstraktan klasa moze imati konstruktor I main
211. Lokalna varijabla nema modifikator pristupa
212. List je interfejs tako da ne mozemo imati List<String> l=new List<String>();

213. Redefinisanje metode run sa public static void run je greska
Modifikator synchronized moze
214. U hes mapi se trebaju redefinisati equals I hes metode
215. Enum kolekcija moze imati konstruktor I varijable
216. Ako pristupamo iz unutrasnje klase lokalnoj promjenljivoj ona mora biti final
217. Unutrasnja klasa se mora prvo definisati pa instancirati
218. Klasa number sadrzi integer
219. Hes set treba da redefinise equals
220.
221. Static nested class nema pristup nestatickim clanovima okruzujuce klase
222. .

```
223.  class Foo
224.  {
225.      class Bar{ }
226.  }
227.  class Test
228.  {
229.      public static void main (String [] args)
230.      {
231.          Foo f = new Foo();
232.          /* Line 10: Missing statement ? */
233.      }
234.  }
```

235. Foo.Bar b = f.new Bar();
236. Instancirali smo Bar