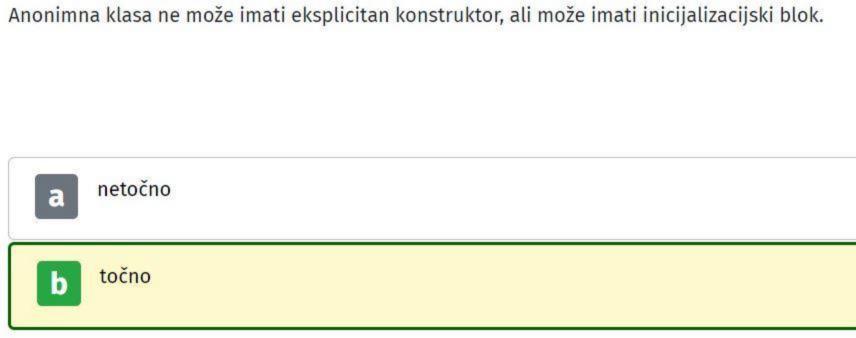


| Anonimn | e klase definiraju novi doseg, odnosno referenca this unutar metode anonimne klase se odnosi na primjerak anonimne klase. |
|---------|---|
| a | točno |
| Ь | netočno |



```
Sto ispisuje program?
class Student {
     String id, name;
     public Student(String i, String n) {
         id = i;
         name = n;
 public class Main {
     public static void main(String[] args) {
         Student pero1 = new Student("123456", "Pero");
         Student pero2 = new Student("123456", "Pero");
         System.out.print(perol.equals(pero2) + ", ");
          System.out.println(perol.equals(perol));
       false, true
       Program je sintaksno ispravan, ali baca iznimku
```

```
Sto ispisuje program?
class Student {
     String id, name;
     public Student(String i, String n) {
          id = i;
          name = n:
  public class Main {
      public static void main(String[] args) {
           List <Student> studentsLst = new ArrayList <Student> ():
           Student pero = new Student("123456", "Pero");
           studentsLst.add(new Student("123450", "Ivo"));
           studentsLst.add(pero);
           System.out.print(studentsLst.contains(pero) + ". ");
           System.out.println(studentsLst.contains(new Student("123450", "Ivo")));
             true, false
```

```
Što ispisuje program?
class Student {
    String id, name:
    public Student(String i, String n) {
        id = i:
        name = n:
    @Override
    public boolean equals(Object obj) {
         if(!(obj instanceof Student))
            return false:
        return this.id.equals(((Student)obj).id);
public class Main {
    public static void main(String[] args) {
        List <Student> studentsLst = new ArrayList <Student> ():
        Student pero = new Student("123456", "Pero");
         studentsLst.add(new Student("123450", "Ivo"));
         studentsLst.add(pero);
         System.out.print(studentsLst.contains(pero) + ". ");
        System.out.println(studentsLst.contains(new Student("123450", "Ivo"))):
    3
         true, true
         falco truo
```

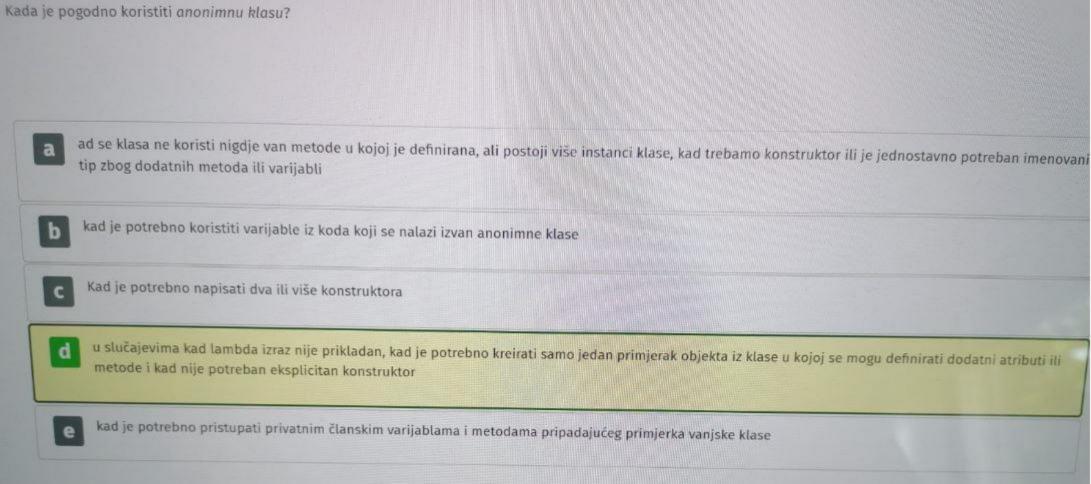
```
Što ispisuje program?
class Student {
    String id, name;
    public Student(String i, String n) {
        id = i; name = n;
     @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof Student)) return false;
        return this.id.equals(((Student)obj).id);
    @Override public int hashCode() {
        return this.id.hashCode():
public class Main {
    public static void main(String[] args) {
        Set<Student> students = new HashSet <Student> ();
        Student pero = new Student("123456", "Pero");
        students.add(new Student("123450", "Ivo"));
        students.add(pero);
        System.out.print(students.contains(pero) + ", ");
        System.out.println(students.contains(new Student("123450", "Ivo")));
    }
         Program je sintaksno ispravan, ali baca iznimku
         false, false
         false, true
         true, true
```

Ako vlastita klasa Student ne implementira nijedno sučelje niti je eksplicitno izvedena iz neke druge klase te u programu postoji sljedeći odsječak koda Student s = new Student("Horvat", "Hrvoje", "19923492"); Set<Student> students = new TreeSet<>(); students.add(s); tada će kompajler automatski stvoriti prirodni komparator za klasu Student

- b elementi u skupu biti složeni u listu umjesto u stablu
- program vršiti usporedbu studenata na temelju metode compareTo iz klase Object
- kompajler prijaviti sintaksnu pogrešku
- se prilikom izvođenja programa dogoditi iznimka

| 9 | klasu možemo instancirati sa sljedećim kodom |
|---|--|
| | new MyCollection.MyIterator.iterable() |
| | instanci vanjske klase možemo pristupiti na sljedeći način |
| | MyCollection.this |
| | klasu možemo instancirati sa sljedećim kodom |
| | new MyCollection.MyIterator() |
| 0 | klasu možemo instancirati sljedećim kodom |
| _ | MyCollection collection = new MyCollection(); MyIterator iterator = collection.new MyIterator(); |

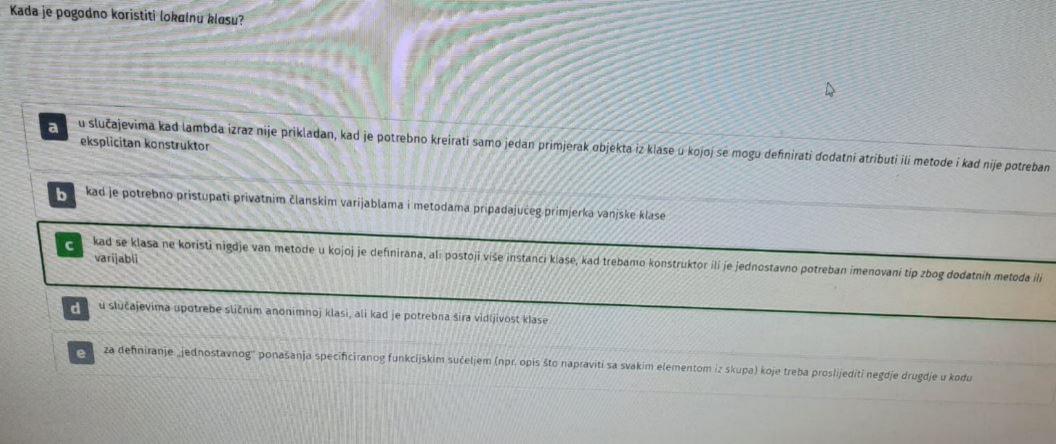
| e od si | jedečih tvrdnji vrijede za anonimne klase? |
|---------|--|
| a | definicija anonimne klase se piše na mjestu stvaranja primjerka u vitičastim zagradama |
| Ь | nemaju ime i najčešće se koriste kad nam je dovoljno stvoriti samo jedan primjerak |
| c | moraju se definirati samostalno tj. ne mogu biti klase koje implementiraju neko sučelje ili nasljedjuju od neke druge klase |
| d | moraju imati eksplicitan konstruktor |



| Što vrijed | li za nestatičke ugnježđene klase: |
|------------|--|
| a | instanca ove klase ne može postojati bez instance vanjske klase |
| b | ovakva klase ne može imati konstruktor |
| С | instanca ovakve klase se može stvoriti na samo jednom mjestu u izvornom kodu |
| d | ovakva klasa nema ime |
| е | instanca ove klase može postojati bez postojanja instance vanjske klase |

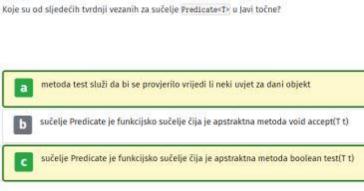
| Što vrije | di za statičke ugnježđene klase: |
|-----------|--|
| a | instanca ovakve klase se može stvoriti na samo jednom mjestu u izvornom kodu |
| Ь | instanca ove klase može postojati bez postojanja instance vanjske klase |
| С | ovakva klasa nema ime |
| d | instanca ove klase ne može postojati bez instance vanjske klase |
| е | ovakva klase ne može imati konstruktor |

| Kada je p | pogodno koristiti <i>unutarnju klasu</i> ? |
|-----------|---|
| a | kad se klasa ne koristi nigdje van metode u kojoj je definirana, ali postoji više instanci klase, kad trebamo konstruktor ili je jednostavno potreban imenovani tip zbog dodatnih metoda ili varijabli |
| b | kad je potrebno pristupati privatnim članskim varijablama i metodama pripadajućeg primjerka vanjske klase |
| С | u slučajevima upotrebe sličnim lokalnoj klasi, ali kad je potrebna šira vidljivost klase |
| d | za definiranje "jednostavnog" ponašanja specificiranog funkcijskim sučeljem (npr. opis što napraviti sa svakim elementom iz skupa) koje treba proslijediti negdje drugdje u kodu |
| е | u slučajevima kad lambda izraz nije prikladan, kad je potrebno kreirati samo jedan primjerak objekta iz klase u kojoj se mogu definirati dodatni atributi ili metode i kad nije potreban eksplicitan konstruktor |



| Kada je j | pogodno koristiti lambda izraz? |
|-----------|--|
| | |
| a | kad se klasa ne koristi nigdje van metode u kojoj je definirana, ali postoji više instanci klase, kad trebamo konstruktor ili je jednostavno potreban imenovani tip zbog dodatnih metoda ili varijabli |
| Ь | kad je potrebno pristupati privatnim članskim varijablama i metodama pripadajućeg primjerka vanjske klase |
| c | u slučajevima kad lambda izraz nije prikladan, kad je potrebno kreirati samo jedan primjerak objekta iz klase u kojoj se mogu definirati dodatni atributi ili metode i kad nije potreban eksplicitan konstruktor |
| d | za definiranje "jednostavnog" ponašanja specificiranog funkcijskim sučeljem (npr. opis što napraviti sa svakim elementom iz skupa) koje treba proslijediti negdje drugdje u kodu |
| е | u slučajevima upotrebe sličnim lokalnoj klasi, ali kad je potrebna šira vidljivost klase |

| Općenito g | govoreći, što od navedenog vrijedi za lambda izraz? |
|------------|---|
| а | u izrazu koji se nalazi desno od strelice "->" definiramo ponašanje metode iz funkcijskog sučelja |
| Ь | u izrazu koji se nalazi lijevo od strelice "->", uz argument obavezno moramo navesti tip argumenta kako bi prevoditelj znao o kojem se funkcijskom sučelju radi |
| С | može zamijeniti anonimnu klasu koja za cilj ima implementirati sučelje s jednom apstraktnom metodom |
| d | može zamijeniti anonimnu klasu koja za cilj ima implementirati određeno funkcijsko sučelje |
| e | lambda izraz, baš kao i anonimna klasa, uvodi novi doseg: this u lambdi se odnosi na primjerak lambde, a ne na primjerak vanjske klase |



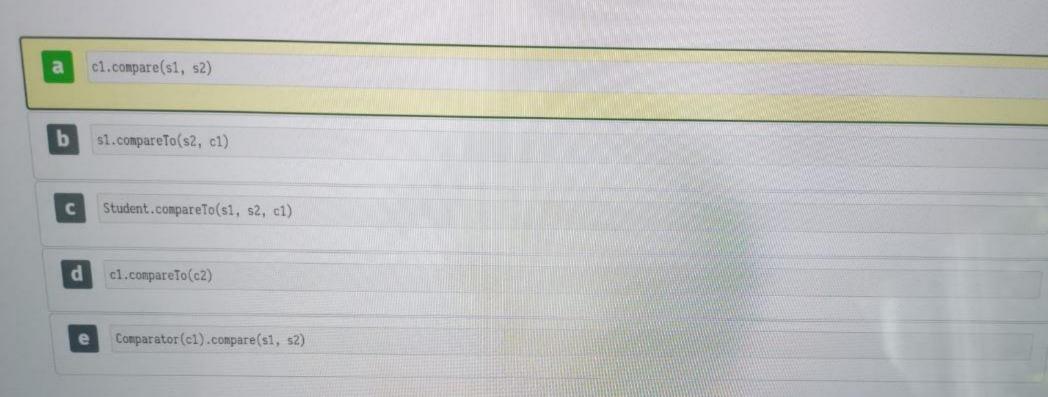
| Općenit | o govoreći, što od navedenog vrijedi za sučelje Predicate <i>?</i> |
|---------|---|
| a | radi se o sučelju koje nije funkcijsko |
| Ь | predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat |
| С | za razliku od sučelja Consumer <t>, sučelje Predicate<t> se od Jave 8 nužno mora implementirati isključivo lambda izrazom</t></t> |
| d | propisuje postojanje metode boolean test(T t) |
| е | propisuje postojanje metode Stream <t> filter(T t)</t> |

| Z | a što na | ım služe predikati (objekti primjerci klasa koje implementiraju sučelje Predicate <t>) u Javi?</t> |
|---|----------|--|
| | а | smanjuju nepotrebno dupliciranje koda u slučajevima kada je potrebno istu metodu izvršiti, ali uz provjeru različitih uvjeta |
| | ь | predikati služe za implementaciju radnji koje objekt može izvršavati |
| | c | implementacija predikata omogućuje definiranje uvjeta za određeni objekt i ovisno o tome vraća istinu ili laž |

| Općenito | govoreći, što od navedenog vrijedi za sučelje Consumer <t>?</t> |
|----------|---|
| | |
| a | za razliku od sučelja <u>Predicate<t></t></u> , sučelje <u>Consumer<t></t></u> se od Jave 8 nužno mora implementirati isključivo lambda izrazom |
| Ь | predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat |
| С | propisuje postojanje metode T remap(T t) |
| d | radi se o sučelju koje nije funkcijsko |
| е | propisuje postojanje metode boolean test(T t) |

Neka klasa Student ne implementira nijedno sučelje niti je eksplicitno izvedena iz neke druge klase te neka je definirana klasa StudentComparator takva da implementira Comparator Student>.

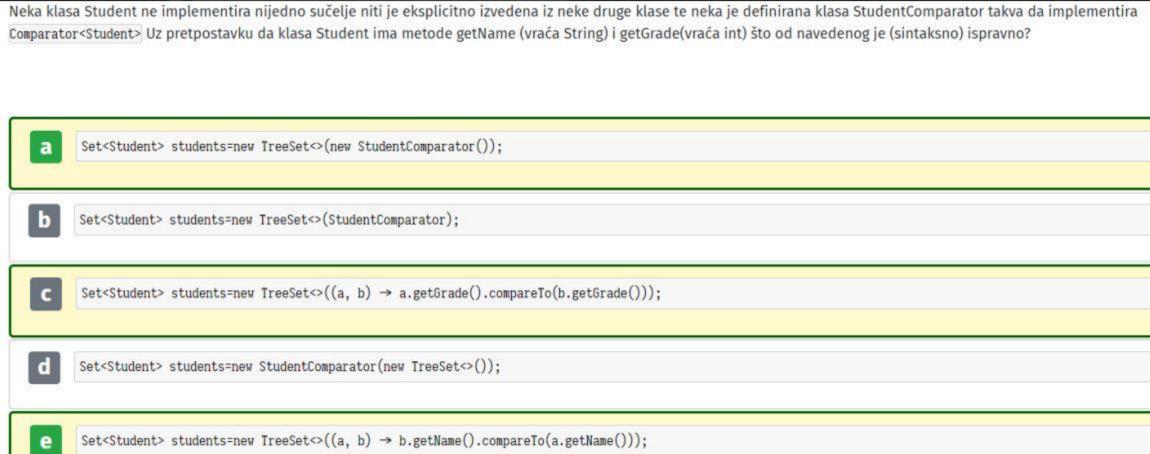
Ako su s1 i s2 tipa Student, c1 i c2 tipa StudentComparator, što od navedenog je (sintaksno) ispravno?



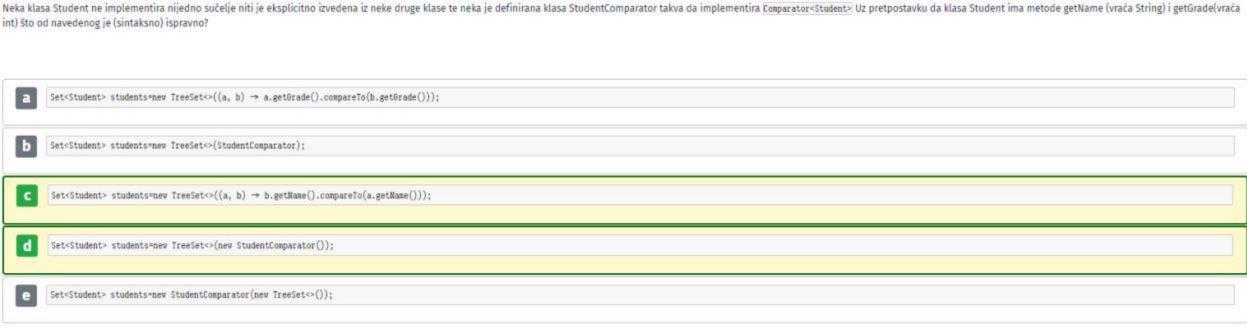
| s1 i s2 tipa StudentŠto je od navedenog (sintaksno) ispravno? |
|---|
| Student.compareTo(s1, s2) |
| Comparable.compareTo(s1, s2) |
| Comparable <student>.compareTo(s1, s2)</student> |
| s2.compareTo(s1) |
| s1.compareTo(s1) |
| |

| pub | public class Kruska implements Comparable <jabuka> ada bi klasa Kruska morala imati metodu</jabuka> | | |
|-----------|---|--|--|
| tada bi l | | | |
| a | int compareTo(Kruska k) | | |
| b | int compareTo(Jabuka j, Kruska k) | | |
| С | int compareTo(Kruska k, Jabuka j) | | |
| d | int compareTo(Jabuka j) | | |
| е | Kompajler ne bi dopustio da klasa Kruska implementira Comparable <jabuka></jabuka> | | |

| Prirodni | poredak, odnosno prirodni komparator za neku klasu T definira se |
|----------|--|
| a | definiranjem nekog komparatora koji se svodi na usporedbu članskih varijabli |
| b | pisanjem statičke metode comparable(T first, T second) |
| С | nadjačavanjem metode compareTo iz klase Object |
| d | implementacijom sučelja Comparable i metode compareTo(T first, T second) |
| е | implementacijom sučelja Comparable i metode compareTo(T other) |



| | sa Student ne implementira nijedno sučelje niti je eksplicitno izvedena iz neke druge klase te neka je definirana klasa StudentComparator takva da implementira Comparator < Student> Uz pretpostavku da klasa Student ima metode getName (vraća String) i getGrade(vraća od navedenog je (sintaksno) ispravno? |
|---|---|
| a | Set <student> students=new TreeSet<>((other) → this.getName().compareTo(other.getName()));</student> |
| Ь | Set <student> students*new TreeSet<>(a.getBrade, a.getName);</student> |
| С | Set <student> students=new TreeSet<>((a, b) → a.getGrade() - b.getGrade());</student> |
| d | Set <student> students=new TreeSet<>(new StudentComparator());</student> |
| е | Set <student> studentSomew StudentComparator(new IreeSet<>());</student> |



Koje su tipične naredbe za početak metode equals koja je nadjačana u klasi Student?

return false;

Novi kolekcijski tok nad (primjerice) listom može se stvoriti pozivom metode

- a iterable()
- b toArray()
- stream()
- d forEach()
- e toList()

| Koji od | navedenih potpisa su potpisi metoda equals i hashCode iz klase java.lang.Obje | ct? |
|---------|---|-----|
| a | boolean equals(Object obj1, Object obj2) | |
| Ь | int hashCode(Object obj) | |
| С | boolean equals(Object obj) | |
| d | int equals(Object obj) | |
| е | int hashCode() | |

Kada se neka klasa nalazi unutar druge klase onda takve klase nazivamo: djecom a ugrađene ukalupljene ugnježđene male klase

| Sučelje Iterable definira metodu za kreiranje iteratora. Koja tvrdnja je ispravna: |
|--|
| iterator pamti svoju poziciju neovisno o drugim stvorenim iteratorima |
| b djelovanje iteratora ovisi o drugim stvorenim iteratorima iz iste kolekcije |
| svaka kolekcija implementira Iterable koji može stvoriti takav iterator |
| d iterator ne smije imati pristup internim elementima kolekcije kroz koju iterira |
| svaka kolekcija prati iteratore koji su iz nje stvoreni i brine se o njihovom dereferenciranju |
| |