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

a kompajler prijaviti sintaksnu pogrešku

b program vršiti usporedbu studenata na temelju metode compareTo iz klase Object

c se prilikom izvođenja programa dogoditi iznimka

Anonimne klase definiraju novi doseg, odnosno referenca this unutar metode anonimne klase se odnosi na primjerak anonimne klase.

a netočno

b točno

Općenito govoreći, što od navedenog vrijedi za sučelje `Consumer<T>`?

a propisuje postojanje metode T remap(T t)

b za razliku od sučelja `Predicate<T>`, sučelje `Consumer<T>` se od Jave 8 nužno mora implementirati isključivo lambda izrazom

c predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat

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?

a `c1.compare(s1, s2)`

Za što nam služe predikati (objekti primjerci klasa koje implementiraju sučelje `Predicate<T>`) u Javi?

a smanjuju nepotrebno dupliciranje koda u slučajevima kada je potrebno istu metodu izvršiti, ali uz provjeru različitih uvjeta

b implementacija predikata omogućuje definiranje uvjeta za određeni objekt i ovisno o tome vraća istinu ili laž

Kada je pogodno koristiti *lokalnu klasu*?

a za definiranje „jednostavnog" ponašanja specificiranog funkcijskim sučeljem (npr. opis što napraviti sa svakim elementom iz skupa) koje treba proslijediti negdje drugdje u kodu

b 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

Funkcijska sučelja (engl. functional interface) su ona sučelja koja imaju samo jednu apstraktnu metodu koju je potrebno implementirati.

a netočno

b točno

Neka je početak definiracije klase Student

```
public class Student implements Comparable<Student>
```

te neka su s1 i s2 tipa StudentŠto je od navedenog (sintaksno) ispravno?

a `Comparable.compareTo(s1, s2)`

b `Student.compareTo(s1, s2)`

c `s2.compareTo(s1)`

d `Comparable<Student>.compareTo(s1, s2)`

e `s1.compareTo(s1)`

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>` Uz pretpostavku da klasa Student ima metode getName (vraća String) i getGrade(vraća int) što od navedenog je (sintaksno) ispravno?

a
```
Set<Student> students=new TreeSet<>((a, b) → b.getName().compareTo(a.getName()));
```

b
```
Set<Student> students=new StudentComparator(new TreeSet<>());
```

c
```
Set<Student> students=new TreeSet<>(new StudentComparator());
```

Što 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(pero1.equals(pero2) + ", ");
        System.out.println(pero1.equals(pero1));
    }
}
```

a true, false

b true, true

c false, false

d Program je sintaksno ispravan, ali baca iznimku

e false, true

Kada je pogodno koristiti *lokalnu klasu*?

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 me*
varijabli

Kada je pogodno koristiti *lambda izraz*?

a kad je potrebno pristupati privatnim članskim varijablama i metodama pripadajućeg primjerka vanjske klase

b 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 dodat varijabli

c za definiranje „jednostavnog" ponašanja specificiranog funkcijskim sučeljem (npr. opis što napraviti sa svakim elementom iz skupa) koje treba proslijediti negdje drugdje u k

Koje od sljedećih tvrdnji vrijede za anonimne klase?

a moraju se definirati samostalno tj. ne mogu biti klase koje implementiraju neko sučelje ili nasljedjuju od neke druge klase

b definicija anonimne klase se piše na mjestu stvaranja primjerka u vitičastim zagradama

c nemaju ime i najčešće se koriste kad nam je dovoljno stvoriti samo jedan primjerak

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

a

```
@Override
public boolean equals(Object obj) {
    if (obj == null) return false;
    if (!(obj instanceof Student))
        return false;
    // ...
```

Neka je početak definiracije klase Student

```
public class Student implements Comparable<Student>
```

te neka su s1 i s2 tipa StudentŠto je od navedenog (sintaksno) ispravno?

a `s1.compareTo(s1)`

b `Comparable.compareTo(s1, s2)`

c `s2.compareTo(s1)`

Prirodni poredak, odnosno prirodni komparator za neku klasu T definira se

a definiranjem nekog komparatora koji se svodi na usporedbu članskih varijabli

b implementacijom sučelja Comparable i metode compareTo(T first, T second)

c nadjačavanjem metode compareTo iz klase Object

d implementacijom sučelja Comparable i metode compareTo(T other)

Neka je početak definiracije klase Student

```
public class Student implements Comparable<Student>
```

te neka su s1 i s2 tipa StudentŠto je od navedenog (sintaksno) ispravno?

a `Comparable.compareTo(s1, s2)`

b `Student.compareTo(s1, s2)`

c `s2.compareTo(s1)`

d `Comparable<Student>.compareTo(s1, s2)`

e `s1.compareTo(s1)`

Koji od navedenih potpisa su potpisi metoda equals i hashCode iz klase java.lang.Object?

a `int equals(Object obj)`

b `int hashCode(Object obj)`

c `boolean equals(Object obj1, Object obj2)`

d `int hashCode()`

e `boolean equals(Object obj)`

Sučelje Iterable definira metodu za kreiranje iteratora. Koja tvrdnja je ispravna:

a iterator pamti svoju poziciju neovisno o drugim stvorenim iteratorima

b djelovanje iteratora ovisi o drugim stvorenim iteratorima iz iste kolekcije

c svaka kolekcija implementira Iterable koji može stvoriti takav iterator

Kada je pogodno koristiti *anonimnu klasu*?

a kad je potrebno pristupati privatnim članskim varijablama i metodama pripadajućeg primjerka vanjske klase

b ad 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

c kad je potrebno koristiti varijable iz koda koji se nalazi izvan anonimne klase

d Kad je potrebno napisati dva ili više konstruktora

e 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

Općenito govoreći, što od navedenog vrijedi za lambda izraz?

a u izrazu koji se nalazi desno od strelice „->“ definiramo ponašanje metode iz funkcijskog sučelja

b 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

c 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

Općenito govoreći, što od navedenog vrijedi za sučelje `Consumer<T>`?

a propisuje postojanje metode boolean test(T t)

b za razliku od sučelja `Predicate<T>`, sučelje `Consumer<T>` se od Jave 8 nužno mora implementirati isključivo lambda izrazom

c propisuje postojanje metode T remap(T t)

d predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat

Što 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(pero1.equals(pero2) + ", ");
        System.out.println(pero1.equals(pero1));
    }
}
```

a false, true

Što vrijedi za statičke ugnježđene klase:

a ovakva klase ne može imati konstruktor

b instanca ovakve klase se može stvoriti na samo jednom mjestu u izvornom kodu

c instanca ove klase ne može postojati bez instance vanjske klase

d ovakva klasa nema ime

e instanca ove klase može postojati bez postojanja instance vanjske klase

Ako bi klasa Kruska bila definirana s

```
public class Kruska implements Comparable<Jabuka>
```

tada bi klasa Kruska morala imati metodu

a Kompajler ne bi dopustio da klasa Kruska implementira `Comparable<Jabuka>`

b `int compareTo(Jabuka j)`

Prirodni poredak, odnosno prirodni komparator za neku klasu T definira se

a definiranjem nekog komparatora koji se svodi na usporedbu članskih varijabli

b implementacijom sučelja Comparable i metode compareTo(T other)

c nadjačavanjem metode compareTo iz klase Object

d pisanjem statičke metode comparable(T first, T second)

e implementacijom sučelja Comparable i metode compareTo(T first, T second)

Općenito govoreći, što od navedenog vrijedi za sučelje `Predicate<T>`?

a za razliku od sučelja `Consumer<T>`, sučelje `Predicate<T>` se od Jave 8 nužno mora implementirati isključivo lambda izrazom

b propisuje postojanje metode `Stream<T> filter(T t)`

c radi se o sučelju koje **nije** funkcijsko

d predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat

e propisuje postojanje metode boolean test(T t)

Za što nam služe predikati (objekti primjerci klasa koje implementiraju sučelje `Predicate<T>`) u Javi?

a smanjuju nepotrebno dupliciranje koda u slučajevima kada je potrebno istu metodu izvršiti, ali uz provjeru različitih uvjeta

b implementacija predikata omogućuje definiranje uvjeta za određeni objekt i ovisno o tome vraća istinu ili laž

Ako implementiramo iterator pomoću statičke ugnježđene klase (MyIterator) koja se nalazi u klasi MyCollection onda za tu klasu vrijedi:

a klasu možemo instancirati sljedećim kodom

```
MyCollection collection = new MyCollection();  MyIterator iterator = collection.new MyIterator(...);
```

b klasu možemo instancirati sa sljedećim kodom

```
new MyCollection.MyIterator(...)
```

c kroz konstruktor takve klase moramo poslati referencu na vanjsku klasu te se takva referenca mora spremiti u atribut

Funkcijska sučelja (engl. functional interface) su ona sučelja koja imaju samo jednu apstraktnu metodu koju je potrebno implementirati.

a netočno

b točno

Koji od navedenih potpisa su potpisi metoda equals i hashCode iz klase java.lang.Object?

a `int hashCode()`

b `int hashCode(Object obj)`

c `boolean equals(Object obj1, Object obj2)`

d `int equals(Object obj)`

e `boolean equals(Object obj)`