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

a točno

b netočno

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

a točno

b netočno

Anonimna klasa ne može imati eksplicitan konstruktor, ali može imati inicijalizacijski blok.

a netočno

b točno

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

c true, false

d false, true

e Program je sintaksno ispravan, ali baca iznimku

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) {
        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")));
    }
}
```

a true, false

b false, 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")));
    }
}
```

a true, true

b false, true

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")));
    }
}
```

a Program je sintaksno ispravan, ali baca iznimku

b false, false

c false, true

d 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

a kompajler automatski stvoriti prirodni komparator za klasu Student

b elementi u skupu biti složeni u listu umjesto u stablu

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

d kompajler prijaviti sintaksnu pogrešku

e se prilikom izvođenja programa dogoditi iznimka

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

a klasu možemo instancirati sa sljedećim kodom

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

b instanci vanjske klase možemo pristupiti na sljedeći način

```
MyCollection.this
```

c klasu možemo instancirati sa sljedećim kodom

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

d klasu možemo instancirati sljedećim kodom

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

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

Koje od sljedećih tvrdnji vrijede za anonimne klase?

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

b 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

Kada je pogodno koristiti *anonimnu klasu*?

a 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

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

c Kad je potrebno napisati dva ili više konstruktora

d 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

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

Što vrijedi za nestatičke ugnježđene klase:

a instanca ove klase ne može postojati bez instance vanjske klase

b ovakva klase ne može imati konstruktor

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

d ovakva klasa nema ime

e instanca ove klase može postojati bez postojanja instance vanjske klase

Što vrijedi za statičke ugnježđene klase:

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

b instanca ove klase može postojati bez postojanja instance vanjske klase

c ovakva klasa nema ime

d instanca ove klase ne može postojati bez instance vanjske klase

e ovakva klase ne može imati konstruktor

Kada je pogodno koristiti *unutarnju 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 metoda ili varijabli

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

c 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

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

Kada je pogodno koristiti *lokalnu klasu*?

a 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

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

c 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

d u slučajevima upotrebe sličnim anonimnoj klasi, ali kad je potrebna šira vidljivost klase

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

Kada je 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

b 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

e u slučajevima upotrebe sličnim lokalnoj klasi, ali kad je potrebna šira vidljivost klase

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

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

Koje su od sljedećih tvrdnji vezanih za sučelje `Predicate<T>` u Javi točne?

a metoda test služi da bi se provjerilo vrijedi li neki uvjet za dani objekt

b sučelje Predicate je funkcijsko sučelje čija je apstraktna metoda void accept(T t)

c sučelje Predicate je funkcijsko sučelje čija je apstraktna metoda boolean test(T t)

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

a. radi se o sučelju koje **nije** funkcijsko

b. predstavlja operaciju koja prihvaća jedan ulazni argument i ne vraća rezultat

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

d. propisuje postojanje metode boolean test(T t)

e. propisuje postojanje metode `Stream<T> filter(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 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>`?

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

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

c propisuje postojanje metode T remap(T t)

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

e 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?

a `c1.compare(s1, s2)`

b `s1.compareTo(s2, c1)`

c `Student.compareTo(s1, s2, c1)`

d `c1.compareTo(c2)`

e `Comparator(c1).compare(s1, s2)`

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 `Student.compareTo(s1, s2)`

b `Comparable.compareTo(s1, s2)`

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

d `s2.compareTo(s1)`

e `s1.compareTo(s1)`

Ako bi klasa Kruska bila definirana s

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

tada bi klasa Kruska morala imati metodu

a `int compareTo(Kruska k)`

b `int compareTo(Jabuka j, Kruska k)`

c `int compareTo(Kruska k, Jabuka j)`

d `int compareTo(Jabuka j)`

e Kompajler ne bi dopustio da klasa Kruska implementira `Comparable<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)

c nadjačavanjem metode compareTo iz klase Object

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

e implementacijom sučelja Comparable i metode compareTo(T other)

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<>(new StudentComparator());
```

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

c
```
Set<Student> students=new TreeSet<>((a, b) -> a.getGrade().compareTo(b.getGrade()));
```

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

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

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<>((other) -> this.getName().compareTo(other.getName()));
```

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

c
```
Set<Student> students=new TreeSet<>((a, b) -> a.getGrade() -  b.getGrade());
```

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

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

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) -> a.getGrade().compareTo(b.getGrade()));
```

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

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

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

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

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

d

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

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

a iterable()

b toArray()

c stream()

d forEach()

e toList()

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

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

b `int hashCode(Object obj)`

c `boolean equals(Object obj)`

d `int equals(Object obj)`

e `int hashCode()`

Kada se neka klasa nalazi unutar druge klase onda takve klase nazivamo:

a djecom

b ugrađene

c ukalupljene

d ugnježđene

e male klase

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

d iterator ne smije imati pristup internim elementima kolekcije kroz koju iterira

e svaka kolekcija prati iteratore koji su iz nje stvoreni i brine se o njihovom dereferenciranju