

Koju apstraktnu metodu je potrebno nadjačati kod implementacije sučelja `java.util.function.Predicate<T>?`

**a**

`boolean compare(T first, T second)`

**b**

`boolean test(T t)`

**c**

`Predicate(boolean b)`

**d**

`void test(T t)`

**e**

`int compareTo(T other)`

Koje su metode definirane sučeljem `java.lang.Iterator<T>`?

**a** `boolean hasNext();`

**b** `Iterator<T> iterator();`

**c** `remove();`

**d** `T next();`

Koje su metode definirane sučeljem `java.lang.Iterable<T>`?

**a**

```
Iterator<T> iterator();
```

Dokumentacija sučelja `java.util.Collection<T>` definira pojam opcionalnih metoda. Sto to znači za razrede koji implementiraju to sučelje i neće podržati akcije koje provode te metode?

**a**

Ti razredi ne smiju definirati te metode tako da njihova implementacija nigdje ne postoji.

**b**

Ti razredi trebaju imati implementirane takve metode na način da u tijelu istih izazovu iznimku `UnsupportedOperationException`.

**c**

Implementacije tih metoda u razredu treba označiti posebnom anotacijom tako da kompajler odbije prevesti kod koji bi ih pozivao.

**d**

Ti razredi trebaju implementirati te metode ali ih označiti kao privatne (modifikator `private`) tako da ih korisnik ne može pozivati. Takve metode razred obavezno mora označiti ključnom riječi `abstract`.

**e**

Takve metode razred obavezno mora označiti ključnom riječi `abstract`.



Dokumentacija sučelja `java.util.Collection` traži od razreda koji implementiraju to sučelje (ili neko iz njega izvedeno) da bi trebali podržati određen broj konstruktora. Od ponuđenih, o kojima se radi?

- a** konstruktor koji prima referencu na neku drugu kolekciju te u konstruiranu kolekciju kopira sve elemente iz predane kolekcije
- b** konstruktor koji prima referencu na dvije kolekcije te u konstruiranu kolekciju kopira sve elemente najprije iz prve a potom iz druge kolekcije
- c** konstruktor koji prima referencu na jedan `Iterable` te u konstruiranu kolekciju dodaje sve elemente koje vrati stvoreni iterator
- d** konstruktor koji prima referencu na jedan `Iterator` te u konstruiranu kolekciju dodaje sve elemente koje iterator vrati
- e** konstruktor koji ne prima niti jedan argument i stvara praznu kolekciju

Što od ponuđenoga vrijedi za sučelje java.util.Collection?

**a**

`Iterable<T> extends Collection<T>`

**b**

`Iterator<T> extends Collection<T>`

**c**

`Collection<T> extends Iterable<T>`

**d**

`Collection<T> extends Iterator<T>`

Što od navedenog je istina za `java.util.Collections`?

**a**

`java.util.Collections` sadrži samo statičke metode

**b**

To je sučelje

**c**

To je statička metoda

**d**

To je metoda

**e**

To je klasa

Koja izjava vrijedi za `java.util.LinkedList`:



lista elemente pohranjuje dinamički alocirajući nove čvorove za dvostruko povezanu listu



Za I/O tokove u Javi vrijedi:

**a**

Tokovi su jednosmjerni

**b**

Klijent u izlazni tok može isključivo zapisivati

**c**

Klijent iz ulaznog toka može isključivo čitati podatak po podatak, ne više njih

**d**

Tokovi se mogu ulančati korištenjem dekoratora



Ispravan način definiranja toka podataka iz datoteke je:



**a**

```
tok = new FileInputStream(<ime_datoteke>)
```

**b**

```
tok = new InputStream(<ime_datoteke>)
```

**c**

```
tok = new OutputStream(<path_do_datoteke>)
```

**d**

```
tok = new File(<ime_datoteke>, Mode.Input)
```

Ispravan način definiranja toka podataka iz datoteke je:

**a**

```
tok = new OutputStream(<path_do_datoteke>)
```

**b**

```
tok = new InputStream(<path_do_datoteke>)
```

**c**

```
tok = Files.newInputStream(<path_do_datoteke>)
```

**d**

```
tok = new File(<path_do_datoteke>, Mode.Output)
```

Označite metode sučelja FileVisitor

**a**

visitDirectory

**b**

postVisitDirectory

**c**

preVisitDirectory

**d**

visitFile

**e**

visitFileFailed

Znakovni tokovi unutar paketa java.io modelirani su apstraktnim razredima

**a**

Listeners

**b**

Charset

**c**

Writer

**d**

Reader

Metode reverse, shuffle i sort dio su klase ili sučelja

**a**

Collection

**b**

Set

**c**

List

**d**

Map

**e**

Collections



dogadja se to u jesenjoj noci kada pada kestenje po asfaltu i kada se cuju psi u daljini

Ako riječi dodajemo u LinkedHashSet redoslijedom kojim su navedene u gornjoj rečenici, što ćemo dobiti kad sadržaj kolekcije ispišemo?

- a** dogadja se to u jesenjoj noci kada pada kestenje po asfaltu i cuju psi daljini
- b** asfaltu cuju daljini dogadja i jesenjoj kada kada kestenje noci pada po psi se se to u u
- c** asfaltu cuju daljini dogadja i jesenjoj kada kestenje noci pada po psi se to u
- d** se u noci to jesenjoj kestenje po cuju pada psi dogadja asfaltu daljini i kada

*dogadja se to u jesenjoj noci kada pada kestenje po asfaltu i kada se cuju psi u daljini*

Ako riječi dodajemo u TreeSet redoslijedom kojim su navedene u gornjoj rečenici, što ćemo dobiti kad sadržaj kolekcije ispišemo?

**a** asfaltu čuju daljini dogadja i jesenjoj kada kada kestenje noci pada po psi se se to u u

**b** asfaltu cuju daljini dogadja i jesenjoj kada kestenje noci pada po psi se to u

Listu možemo stvoriti pozivom:

**a** `new ArrayList<>();`

**b** `new List();`

**c** `Arrays.asList("A", "B", "C");`

**d** `new LinkedList();`

**e** `Arrays.asList(1,2,3);`

Lista je uređena kolekcija koja



ne može sadržavati duplikate



može sadržavati duplikate

Izvor i ponor okteta modelirani su:

**a**

apstraktnim klasama `InputStream` i `OutputStream`

**b**

sučeljima `Input` i `Output`

**c**

objektima `System.in` i `System.out`

**d**

apstraktnim klasama `InputStream`, `OutputStream` i `ChannelStream`

Mostovi između znakovnih tokova i tokova okteta su



FileWriter



FileReader



InputStreamReader



OutputStreamWriter



Most između znakovnih tokova i tokova okteta je

**a**

StringReader

**b**

StringStream

**c**

InputStreamReader

**d**

FileReader

Koji od ponuđenih naziva predstavljaju klase koje implementiraju sučelje `java.util.Set`, a koje pripadaju Javinom okviru kolekcija?

**a** HashSet

**b** ArraySet

**c** LinkedSet

**d** TreeSet

Koji od ponuđenih naziva predstavljaju klase koje implementiraju sučelje `java.util.Set`, a koje pripadaju Javinom okviru kolekcija?

**a** `LinkedHashSet`

**b** `TreeSet`

**c** `HashSetMap`

**d** `LinkedMapSet`

Implementacija sučelja `java.util.Set` iz Javinog okvira kolekcija koja osigurava da su elementi sortirani je:

**a**

HashSet

**b**

ComparableSet

**c**

LinkedHashSet

**d**

TreeSet

Implementacija sučelja `java.util.Set` iz Javinog okvira kolekcija koja osigurava da su elementi poredani redoslijedom dodavanja je:

**a**

TreeSet

**b**

HashSet

**c**

OrderedSet

**d**

LinkedHashSet

Javin okvir kolekcija nudi sljedeće implementacije sučelja `java.util.List`:

**a** `MapList`

**b** `TreeList`

**c** `ArrayList`

**d** `LinkedList`

**e** `HashList`



Kolekcija Map je parametrizirana s tipom ključa i vrijednosti. Što može biti korišteno kao ključ i vrijednost?

**a** samo klase koje nasljeđuje klasu String ili objektni omotači primitivnih tipova

**b** bilo koja klasa

**c** primitivni tip

**d** bilo koje sučelje

**e** samo klase koji su objektni omotači primitivnih tipova

Koje su karakteristike kolekcije Map?

**a**

svaki ključ ima pridruženu jednu vrijednost

**b**

mapa može pohranjivati više istih ključeva

**c**

mapa **ne može** pohranjivati više istih ključeva

**d**

više ključeva **ne smije** imati istu vrijednost

**e**

više ključeva može imati istu vrijednost

Što od navedenog je ispravno ako je *map* tipa `Map<String, Integer>`?

**a**

```
for(String entry : map) {  
    // radi nešto  
}
```

**b**

```
for(String entry : map.keySet()) {  
    // radi nešto  
}
```

Što od navedenog je ispravno ako je *map* tipa `Map<String, Integer>`?

**a**

```
for(var entry : map) {  
    // radi nešto  
}
```

**b**

```
for(Map.Entry<String, Integer> entry : map.keySet()) {  
    // radi nešto  
}
```

**c**

```
for(Map.Entry<String, Integer> entry : map.values()) {  
    // radi nešto  
}
```

**d**

```
for(Map.Entry<String, Integer> entry : map) {  
    // radi nešto  
}
```

**e**

```
for(Map.Entry<String, Integer> entry : map.entrySet()) {  
    // radi nešto  
}
```

Što sve od navedenog je ispravno ako je *map* tipa `Map<String, Map<String, Integer>>`?

**a**

```
for(Map.Entry<String, Map.Entry<String, Integer>> entry : map.entrySet()) {  
    // radi nešto  
}
```

**b**

```
for(Map.Entry<String, Map<String, Integer>> entry : map.entrySet()) {  
    // radi nešto  
}
```

**c**

```
for(Map<String, Integer> entry : map.keySet()) {  
    // radi nešto  
}
```

**d**

```
for(Map.Entry<Map<String, Integer>> entry : map.values()) {  
    // radi nešto  
}
```

**e**

```
for(Map<String, Integer> entry : map.values()) {  
    // radi nešto  
}
```



Ako je mapa definirana s `Map<String,Integer> map` ispravno inicijalizirana te je prazna u trenutku izvođenja sljedećih naredbi,

```
map.put("Ana", 5);  
map.put("Ana", 4);  
System.out.println(map.size());  
System.out.println(map.get("Ana"));
```

što će se ispisati na ekranu?



1  
4



Ako je skup definiran s `Set<String> set` ispravno inicijaliziran te je prazan u trenutku izvođenja sljedećih naredbi,

```
System.out.println(set.add("Ana"));  
System.out.println(set.add("Ana"));
```

što će se ispisati na ekranu?

**a**

1  
2

**b**

false  
Ana

**c**

false  
true

**d**

true  
false

Metoda vratiKolekciju() vraća ne-null referencu na neku nepromjenjivu kolekciju. Što je ponuđenoga potrebno upisati na mjesto označeno podvlakama u kodu kako metoda m ne bi svojem pozivatelju propagirala iznimku nastalu uslijed poziva metode add u bloku try?

```
public void m() {  
    try {  
        Collection<String> kolekcija = vratiKolekciju();  
        kolekcija.add("Pero");  
        System.out.println(kolekcija.contains("Pero"));  
    }  
    catch(-----) { }  
}
```

**a**

`IndexOutOfBoundsException ex`

**b**

`InvalidArgumentException ex`

**c**

`NullPointerException ex`

**d**

`UnsupportedOperationException ex`

Metoda vratiKolekciju() vraća referencu na neku kolekciju. Što treba upisati na mjesta označena u kodu kako bi program ispisao sve elemente kolekcije na zaslon?

```
Collection<String> kolekcija = vratiKolekciju();  
for(__(1)__: ____(2)__) {  
    System.out.println(__(3)__);  
}
```

**a**

(1) String str, (2) kolekcija, (3) iterator().next()

**b**

(1) String str, (2) kolekcija.iterator(), (3) str

**c**

(1) String str, (2) kolekcija, (3) iterator()

**d**

(1) String str, (2) kolekcija, (3) str

Napisali smo razred koji implementira sučelje `FilenameFilter`. U tom razredu smo napisali:

```
public boolean accept(File dir, String name) {    --- KOD --- }
```

Što treba upisati na mjesto linije "--- KOD ---" ako želimo provjeriti da li je objekt iz datotečnog sustava za koji se poziva metoda `accept` ujedno i direktorij?

**a**


```
return new File(dir,name).isDirectory();
```

**b**

```
return name.endsWith(".");
```

**c**

```
return !(name.isFile());
```

 `return dir.isDirectory();`



Što od navedenog vrijedi pri preslikavanju primjeraka razreda File i Path:

- a primjerci razreda File mogu se preslikati u primjerke razreda Path koristeći metodu toPath(), a primjerci razreda Path se ne mogu preslikati u primjerke razreda File
- b U Javi ne postoje File i Path, već samo Files i Paths
- c primjerci razreda Path mogu se preslikati u primjerke razreda File koristeći metodu toFile(), a primjerci razreda File se ne mogu preslikati u primjerke razreda Path
- d primjerci razreda File ne mogu se preslikati u primjerke razreda Path, a ni primjerci razreda Path se ne mogu preslikati u primjerke razreda File
- e primjerci razreda File mogu se preslikati u primjerke razreda Path koristeći metodu toPath(), a primjerci razreda Path mogu se preslikati u primjerke razreda File koristeći metodu toFile()**

Ako se za obradu strukture direktorija oslonimo na metodu `Files.walkFileTree(...)`, jedan od objekata koji je potrebno predati može biti objekt koji je (označite sve što je tačno):

**a**

primjerak klase koja implementira `FileVisitor`

**b**

primjerak klase koja implementira `SimpleFileVisito`

**c**

primjerak klase koja nasljeđuje `FileVisitor`

**d**

primjerak klase koja nasljeđuje `SimpleFileVisitor`

Ako se za obradu strukture direktorija oslonimo na metodu `Files.walkFileTree(...)`, označite sve što potrebno predati toj metodi kroz argumente:

- ☒ **a** objekt koji implementira sučelje `FileVisitor`
- ☐ **b** iterator koji vraća datoteke u direktoriju
- ☐ **c** vršni direktorij iz kojeg kreće obilazak
- ☒ **d** objekt koji implementira sučelje `DirectoryStream.Filter`



Pretpostavimo da koristimo računalo s Windows OS-om i da NE POSTOJI datoteka na putanji "d:/tmp/readme.txt". Što će se dogoditi ako pokrenemo program koji sadrži sljedeću liniju koda:

```
File f = new File("d:/tmp/readme.txt");
```

- a** Naredba će se uspješno izvršiti, tj. stvorit će se objekt tipa File bez obzira na to što datoteka ne postoji.

Pretpostavimo da koristimo računalo s Windows OS-om i da NE POSTOJI datoteka na putanji "d:/tmp/readme.txt". Što će se dogoditi ako pokrenemo program koji sadrži sljedeću liniju koda:

```
File f = new File("d:/tmp/readme.txt");
```

- a** Pogreška prevoditelja, ne postoji File konstruktor koji prima samo jedan parametar.
- b** Pogreška prevoditelja, datoteka ne postoji i kod se neće prevesti.
- c** Naredba će se uspješno izvršiti, a na navedenoj putanji će se stvoriti prazna datoteka readme.txt.
- d** Naredba će se uspješno izvršiti, tj. stvorit će se objekt tipa File, ali ne i datoteka readme.txt.
- e** Kod će se prevesti ali dogodit će se pogreška pri izvođenju navedene linije, tj. "File not found exception".



Što sve vrijedi za kolekciju tipa *List*

**a**

može se dohvatiti element na zadanoj poziciji

**b**

kolekcija je nepromjenjiva

**c**

elementi se mogu dodavati samo na kraj

**d**

može se umetnuti element na zadanu poziciju

**e**

može se obrisati element na zadanoj poziciji

Što sve vrijedi za kolekciju tipa *List*

**a**

numeracija pozicija počinje od 1

**b**

može se dohvatiti element na zadanoj poziciji

**c**

numeracija pozicija počinje od 0

**d**

elementi nemaju svoju poziciju unutar liste

**e**

elementi imaju svoju poziciju unutar liste

Ako se kod kreiranja novog `FileInputStream` objekta proslijedi kao parametar ime datoteke koja ne postoji, dogoditi će se:



Bacit će iznimka `FileNotFoundException`

Nad objektima koji su primjerci razreda koji implementira sučelje `DirectoryStream`, koje sve metode sigurno možemo pozvati (jer postoje)?

**a** `hasNext()`

**b** `iterator()`

**c** `close()`