OBJEKTNO-ORIJENTISANI DIZAJN SOFTVERA

Pomoćni materijal za završni ispit

split metoda klase String

- Metoda `split` klase `String` „razbija“ predmetni string na podstringove razdvojene delimiterima (spejs, tab, zarez, Enter...), i vraća niz `String` objekata.

```
String recenica = "Java je moćan jezik.";
String reci[] = recenica.split(" ");
for(String s: reci)
     System.out.println(s);
```

```
Java
je
moćan
jezik.
```

Ispis

- Argument metode `split` može biti regularan izraz, čime se može izvršiti složenije razbijanje na podstringove.

Konverzija stringa u numeričke tipove

```
public class Test {

  public static void main(String[] args) {
    String strInt = "100", strDouble = "67.89";
    int cio = Integer.parseInt(strInt);
    double real = Double.parseDouble(strDouble);
    System.out.printf("Cio broj: %d, realan: %f", cio, real);
  }
}
```

Ispis

```
Cio broj: 100, realan: 67.890000
```

List kolekcija

- `List` je uređena kolekcija koja može sadržati duplikate elemenata.
- Kao kod nizova, indeks prvog elementa liste je 0. Za razliku od nizova, elementima se ne može pristupiti koristeći zagrade `[]`, već preko odgovarajućih metoda (`set` i `get`).
- Pored metoda nasleđenih iz interfejsa `Collection`, interfejs `List` obezbeđuje niz metoda za rad sa elementima liste (preko indeksa), rad sa opsegom elemenata liste, pretragu elemenata itd.
- Nekoliko klasa implementira interfejs `List`, od kojih su najčešće korišćene **`ArrayList`** i **`LinkedList`**.
- `ArrayList` se implementira kao niz promenljive dužine. Stoga su operacije gde se često menja broj elemenata `ArrayList` objekta neefikasne.
- `LinkedList` sa implementira kao povezana lista. Umetanje novih elemenata u sredinu liste, brisanje elemenata liste i slično su značajno efikasniji u odnosu na `ArrayList` objekte.

Neke metode i osobine klase ArrayList

Metoda	Opis
`add`	Dodavanje elementa na kraj kolekcije ili na određenu poziciju.
`clear`	Brisanje kolekcije (uklanjanje svih elemenata).
`contains`	Vraća true ako kolekcija sadrži traženi element i false u suprotnom.
`get`	Vraća element sa specificiranim indeksom.
`set`	Postavlja vrijednost elementu sa specificiranim indeksom.
`indexOf`	Vraća indeks prve pojave specificiranog elementa kolekcije.
`remove`	Uklanja prvu pojavu specificirane vrednosti ili elementa sa specificiranim indeksom.
`size`	Vraća broj elemenata smeštenih u ArrayList kolekciji.
`toArray`	Konvertovanje kolekcije u niz. Ovo se radi da bi se određene operacije sa nizom izvele efikasnije, kao i da bi se konvertovani niz mogao proslediti metodama koje ne rade sa kolekcijama ili obraditi nasleđenim starim kodom koji ne poznaje kolekcije.
`trimToSize`	Smanjenje kapaciteta kolekcije na trenutni broj elemenata.

Klasa ArrayList – Primer

```
import java.util.ArrayList;

public class PrimerArrayList {

  public static void main(String[] args) {
    ArrayList< String > kolString = new ArrayList< String >();

    System.out.printf("Početna veličina: %d\n", kolString.size());

    kolString.add("Prvi");
    kolString.add("Drugi");
    kolString.add(0,"Treći");
    kolString.add(1,"Četvrti");

    System.out.printf("Veličina nakon dodavanja: %d\n", kolString.size());
    System.out.printf("Kolekcija nakon dodavanja: ");
    for(String s: kolString)
      System.out.printf("%s ", s);

    String[] nizString = new String[kolString.size()];
    kolString.toArray(nizString);
    System.out.printf("\nNiz nakon kopiranja kolekcije u niz: ");
    for(String s: nizString)
      System.out.printf("%s ", s);

    // Nastavak na narednom slajdu...
```

```
    System.out.printf("\nU kolekciji %s elementa %s",
        kolString.contains("Treći") ? "ima" : "nema", "Treći");
    System.out.printf("\nU kolekciji %s elementa %s\n",
        kolString.contains("Peti") ? "ima" : "nema", "Peti");

    kolString.remove("Treći");
    System.out.printf("Kolekcija nakon uklanjanja elementa Treći: ");
    for(String s: kolString)
      System.out.printf("%s ", s);
    kolString.remove(0);
    System.out.printf("\nKolekcija nakon uklanjanja elementa sa indeksom 0: ");
    for(String s: kolString)
      System.out.printf("%s ", s);

    kolString.clear();
    System.out.printf("\nVeličina nakon brisanja kolekcije: %d", kolString.size());
  }

}
```

```
Početna veličina: 0
Veličina nakon dodavanja: 4
Kolekcija nakon dodavanja: Treći Četvrti Prvi Drugi
Niz nakon kopiranja kolekcije u niz: Treći Četvrti Prvi Drugi
U kolekciji ima elementa Treći
U kolekciji nema elementa Peti
Kolekcija nakon uklanjanja elementa Treći: Četvrti Prvi Drugi
Kolekcija nakon uklanjanja elementa sa indeksom 0: Prvi Drugi
Veličina nakon brisanja kolekcije: 0
```

Ispis

Korisne metode LinkedList kolekcije

Metoda	Opis
`add`	Dodaje element na kraj LinkedList kolekcije (verzija metode sa jednim parametrom) ili na određenu poziciju (pozicija prvi parametar, element drugi parametar).
`addFirst` `addLast`	Dodaju element na prvu, odnosno zadnju poziciju u listi.
`addAll`	Nadovezuje kolekciju (parametar metode) na predmetnu listu.
`remove`	Uklanja prvu pojavu specificirane vrednosti ili elementa sa specificiranim indeksom. Specificirana vrednost ili indeks su parametri metode.
`removeFirst` `removeLast`	Uklanjaju prvi, odnosno zadnji element liste.
`clear`	Briše kolekciju (uklanja sve elemente).
`contains`	Vraća true ako kolekcija sadrži traženi element i false u suprotnom.
`get`	Vraća element sa specificiranim indeksom.
`indexOf`	Vraća indeks prve pojave specificiranog elementa kolekcije, odnosno -1 ako element ne postoji
`size`	Vraća broj elemenata smeštenih u kolekciji.
`toArray`	Konvertovanje kolekcije u niz.

asList metoda klase Arrays

- **`asList`** metoda klase `Arrays` omogućava da se predmetni niz posmatra kao `List` kolekcija.
- `asList` metoda vraća `List` izgled (eng. *`List` view*) niza, koji se u tom slučaju naziva podržavajući niz (eng. *backing array*).
- Bilo kakva promena `List` izgleda menja podržavajući niz i obrnuto.
- Jedina dozvoljena operacija na `List` izgledu koga vraća metoda `asList` je **`set`**, koja menja vrednost elementa izgleda i odgovarajućeg elementa niza.

Primer za asList

```
import java.util.List;
import java.util.Arrays;

public class asListPodrzavajuciNiz {

   public static void main(String[] args) {
      String imena[] = {"Teodora", "Enis", "Robert", "Marija", "Ana"};

      List< String > lista = Arrays.asList(imena);
      System.out.printf("Početna lista:\n%s\n", lista);

      lista.set(0, "Milutin");
      imena[1] = "Eva";

      System.out.println("Krajnji niz:");
      for(String ime: imena)
         System.out.print(ime + " ");

      System.out.printf("\nKrajnja lista:\n%s", lista);
   }
}
```

Ispis

```
Početna lista:
[Teodora, Enis, Robert, Marija, Ana]
Krajnji niz:
Milutin Eva Robert Marija Ana
Krajnja lista:
[Milutin, Eva, Robert, Marija, Ana]
```

Primer za asList i toArray

```
import java.util.LinkedList;
import java.util.Arrays;

public class AsListToArray {

   public static void main(String[] args) {

      String imena[] = {"Petar", "Milan", "Teodora"};
      LinkedList< String > lista = new LinkedList< String >(Arrays.asList(imena));

      System.out.println("Početna lista:\n" + lista);

      lista.add("Rade");
      lista.add(2, "Vesna");
      lista.addFirst("Marija");

      System.out.printf("Lista nakon dodavanja:\n%s\n", lista);

      imena = lista.toArray(new String[lista.size()]);

      System.out.println("Niz nakon kopiranja liste:");
      for(String ime: imena)
         System.out.print(ime + " ");
   }
}
```

Ispis

```
Početna lista:
[Petar, Milan, Teodora]
Lista nakon dodavanja:
[Marija, Petar, Milan, Vesna, Teodora, Rade]
Niz nakon kopiranja liste:
Marija Petar Milan Vesna Teodora Rade
```

Klasa Collections

- Klasa **`Collections`** obezbeđuje nekoliko efikasnih algoritama za manipulaciju elementima kolekcije. Ovi algoritmi su implementirani kao statičke metode. Najčešće korišćene metode su date u tabeli ispod.

Metoda	Opis
`sort`	Sortiranje elemenata liste.
`binarySearch`	Traženje elementa liste.
`reverse`	Obrtanje redosleda elemenata liste.
`shuffle`	Slučajno raspoređivanje elemenata (mešanje) liste.
`fill`	Upisivanje određene reference u svaki element liste.
`copy`	Kopiranje reference iz jedne liste u drugu.
`min`	Vraća minimalan element kolekcije.
`max`	Vraća maksimalan element kolekcije.
`addAll`	Nadovezivanje elemenata niza na kolekciju.

Primer sa klasom Collections

```
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;
import java.util.Arrays;

public class CollectionsTest {

   public static void main(String[] args) {
      Integer niz[] = {3, 4, 11, 9, 7, 4, 17, 1};
      List< Integer > lista = new ArrayList< Integer >(Arrays.asList(niz));
      System.out.printf("Početna lista:\n%s ", lista);

      System.out.printf("\nMinimum liste %d, maksimum %d.",
           Collections.min(lista), Collections.max(lista));

      Collections.sort(lista);
      System.out.printf("\nLista nakon rast. sortiranja:\n%s", lista);
      Collections.sort(lista, Collections.reverseOrder());
      System.out.printf("\nLista nakon opad. sortiranja:\n%s", lista);

      Collections.shuffle(lista);
      System.out.printf("\nLista nakon mešanja:\n%s", lista);

      Collections.addAll(lista, 5, 13);
      System.out.printf("\nLista nakon dodavanja:\n%s", lista);

      Collections.fill(lista, 7);
      System.out.printf("\nLista nakon zamene elemenata\n%s", lista);
   }
}
```

Ispis

```
Početna lista:
[3, 4, 11, 9, 7, 4, 17, 1]
Minimum liste 1, maksimum 17.


Lista nakon rast. sortiranja:
[1, 3, 4, 4, 7, 9, 11, 17]
Lista nakon opad. sortiranja:
[17, 11, 9, 7, 4, 4, 3, 1]

Lista nakon mešanja:
[9, 11, 3, 1, 7, 4, 17, 4]

Lista nakon dodavanja:
[9, 11, 3, 1, 7, 4, 17, 4, 5, 13]

Lista nakon zamene elemenata
[7, 7, 7, 7, 7, 7, 7, 7, 7, 7]
```

Interfejs Comparator

- **`Comparator`** je generički interfejs iz paketa `java.util` koji se koristi za sortiranje objekata korisničkih klasa. U odnosu na `Comparable` interfejs:
  - omogućava poređenje i sortiranje po više kriterijuma,
  - ne zahteva izmenu originalne klase, odnosno omogućava poređenje za klase čiju realizaciju ne možemo modifikovati.
- `Comparator` interfejs omogućava proizvoljan broj načina sortiranja elemenata predmetnog tipa. Za svaki način sortiranja, potrebno je da definišemo klasu koja implementira generički interfejs `Comparator< Tip >`, u okviru koje ćemo realizovati metodu **`compare`**. Ova metoda ima dva parametra - objekte predmetnog tipa koje poredi, i vraća negativan ceo broj ako je prvi parametar objekat manji od drugog, pozitivan ceo broj ako je prvi parametar veći od drugog, i 0 ako su jednaki.

```
import java.util.Comparator;

public class CovekRastuci implements Comparator< Covek > {

   public int compare(Covek prvi, Covek drugi) {

      double vis1 = prvi.getVisina();    double tez1 = prvi.getTezina();
      double vis2 = drugi.getVisina();   double tez2 = drugi.getTezina();
      if((vis1 == vis2) && (tez1 == tez2))  return 0;
      else if((vis1 > vis2) || ((vis1 == vis2) && (tez1 > tez2)))  return 1;
      else return -1;
   }
}
```

Skupovi. HashSet

- **Skup** (eng. *set*) predstavlja kolekciju jedinstvenih elemenata.
- Postoji nekoliko **`Set`** implementacija, od kojih su najpopularniji **`HashSet`** i **`TreeSet`**.
- `HashSet` smešta elementa u hash tabeli ili hash mapi, dok `TreeSet` smešta elemente u stablu. U TreeSet-u, elementi su sortirani.
- **`HashSet`** je klasa iz paketa `java.util` koja za smeštanje i indeksiranje elemenata koristi hash tabelu.

Metoda	Opis
`add`	Dodaje element (parameter metode) u `HashSet` ukoliko taj element ne postoji.
`addAll`	Nadovezuje elemente niza (drugi parametar) na kolekciju (prvi parametar)
`remove`	Uklanja objekat (parametar metode) iz skupa.
`clear`	Briše sve elemente iz skupa.
`contains`	Proverava da li postoji element (parameter metode) u skupu i vraća `true` ako postoji, odnosno `false` ako ne postoji.
`isEmpty`	Vraća `true` ako je skup prazan, `false` u suprotnom.
`size`	Vraća broj elemenata (kardinalni broj) skupa.

Klasa TreeSet

- `TreeSet` implementira **`SortedSet`** interfejs, koji nasleđuje `Set` interfejs.
- Sve kolekcije koje implementiraju `SortedSet` interfejs imaju sortirane elemente, bilo u prirodnom poretku (npr. rastući redosled za brojeve) ili u poretku koji definiše `Comparator` objekat prosleđen `TreeSet` konstruktoru.
- Osnovne operacije sa elementima (dodavanje, uklanjanje i provera da li element postoji) kod `TreeSet`-a su sporije nego kod `HashSet`-a, ali još uvek mnogo brže nego kod nizova ili povezanih listi.
- Ako `TreeSet` sadrži N elemenata, potrebno je u proseku $\log_2 N$ poređenja za ove osnovne operacije, dok je kod `HashSet`-a složenost ovih operacija konstantna, tj. ne zavisi od broja elemenata.
- `TreeSet` ima četiri konstruktora. Pomenimo 1) konstruktor sa parametrom kolekcijom čije elemente kopira u kreiranu `TreeSet` instancu i sortira prema prirodnom poretku, i 2) konstruktor sa parametrom komparatorom koji definiše poredak elemenata.

Primer sa Set-ovima

```
import java.util.Comparator;

public class KomparatorStringova implements Comparator<String> {
   public int compare(String prvi, String drugi) {
      return prvi.length() - drugi.length();
   }
}
```

Dužina stringa kao kriterijum poređenja

```
import java.util.*;

public class HashSetTreeSet {

   public static void main(String[] args) {
     String nizImena[] = {"Ana", "Marko", "Robert", "Esma", "Dalibor",
                          "Igor", "Ivana", "Marko", "Robert", "Ana"};

     Set< String > hSet = new HashSet< String >(Arrays.asList(nizImena));
     Set< String > tSet1 = new TreeSet< String >(hSet);
     Set< String > tSet2 = new TreeSet< String >(new KomparatorStringova());
     tSet2.addAll(tSet1);

     System.out.printf("HashSet:\n%s", hSet);
     System.out.printf("\nTreeSet (prirodni poredak):\n%s", tSet1);
     System.out.println("\nTreeSet (korisnički definisan poredak):");
     for(String ime: tSet2)
          System.out.print(ime + " ");
   }
}
```

```
HashSet:
[Esma, Igor, Ana, Robert, Marko, Dalibor, Ivana]
TreeSet (prirodni poredak):
[Ana, Dalibor, Esma, Igor, Ivana, Marko, Robert]
TreeSet (korisnički definisan poredak):
Ana Esma Ivana Robert Dalibor
```

Mape

- Za razliku od skupova, koji sadrže samo vrednosti, vrednosti **mape** (eng. *map*) su parovi ključ-vrednost (eng. *key-value*).
- Mapa je objekat koja mapira (preslikava) ključeve u vrednosti. Ključevi u mapi moraju biti jedinstveni, dok odgovarajuće vrednosti ne moraju biti. Ključevi i vrednosti mogu biti proizvoljni referencijski tipovi.

Metoda	Opis
`put`	Dodaje ključ (prvi parameter) i odgovarajuću vrednost (drugi parameter) u mapu. Ukoliko ključ već ima pridruženu vrednost, metoda menja staru vrednost novom. **Primer**: Ako je ključ tipa `String`, a vrednost `Integer`, naredba `mapa.put("Valjevo", 14000)` će dodati ključ `"Valjevo"` sa pripadajućom vrednošću 14000 u mapu.
`putAll`	Kopira sve parove ključ-vrednost iz mape parametra u predmetnu mapu.
`get`	Vraća vrednost koja odgovara ključu (parametar metode), odnosno null ako ne postoji ta vrednost. **Primer**: Naredba `mapa.get("Valjevo")` će vratiti vrednost 14000, nakon naredbe iz primera `put` metode.
`remove`	Uklanja par ključ-vrednost iz mape za prosleđeni ključ (parametar metode) iz mape. Metoda vraća vrednost dodeljenu ključu, odnosno `null` ako postoji predmeti par ključ-vrednost.
`clear`	Briše sve parove ključ-vrednost iz mape.
`containsKey`	Vraća `true` ako mapa sadrži mapiranje sa predmetnim ključem, `false` ako ne sadrži.
`containsValue`	Vraća `true` ako mapa sadrži jedno ili više mapiranja sa predmetnom vrednošću, `false` ako ne sadrži.
`keySet`	Vraća `Set` izgled (eng. *Set view*) ključeva mape. Mapa podržava skup ključeva, tako da se promene na mapi odražavaju na skup i obrnuto.
`values`	Vraća `Collection` izgled (eng. *Collection view*) vrednosti sadržanih u mapi. Mapa podržava kolekciju, tako da se promene na mapi odražavaju na kolekciju i obrnuto.
`entrySet`	Vraća `Set` izgled parova ključ-vrednost sadržanih u mapi. Mapa podržava skup ključeva, tako da se promene na mapi odražavaju na skup i obrnuto.
`isEmpty`	Vraća `true` ako je mapa prazna, `false` u suprotnom.
`size`	Vraća broj parova ključ-vrednost mape.

Mape

- Neke od klasa koje implementiraju interfejs `Map` su **`HashTable`**, **`HashMap`** i **`TreeMap`**. `HashTable` i `HashMap` smeštaju elemente u hash tabelama, dok ih `TreeMap` smešta u stablo.
- Interfejs **`SortedMap`** nasleđuje `Map`, i klase koje implementiraju `SortedMap` imaju sortirane ključeve. Klasa `TreeMap` implementira `SortedMap`.
- `HashTable`, `HashMap` i `TreeMap` su generičke klase čije instance se kreiraju na sledeći način (dajemo primer sa `HashMap`):

```
Map< K, V > map = new HashMap< K, V >();
```

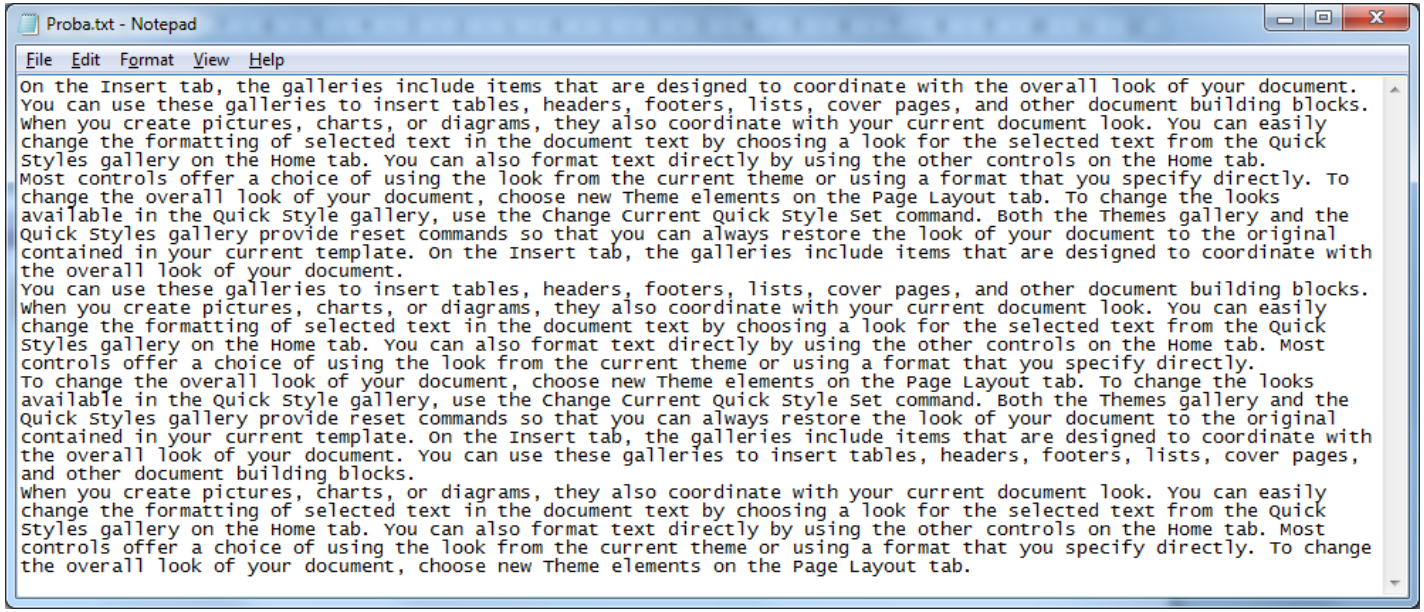
gde `K` i `V` predstavljaju tip ključa i vrednosti, respektivno.

- U nastavku dajemo primer korišćenja mape za brojanje pojava svake reči u datom tekstualnom fajlu.

Primer sa TreeMap

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
import java.util.*;

public class RadSaMapom {

  public static void main(String[] args) {
    List< String > listaLinija = null;

    try {
      Path putanja = Paths.get("C:\\Temp\\Tekst.txt");
      listaLinija = Files.readAllLines(putanja, StandardCharsets.UTF_8); }
    catch(IOException e) {
      System.err.println("Greška prilikom čitanja fajla.");
      System.exit(1); }

    Map< String, Integer > mapa = new TreeMap< String, Integer >();

    for(String linija: listaLinija){
      linija = linija.toLowerCase();
      String reci[] = linija.split("[ ,.;!?-]");
      for(String rec: reci){
        if(rec.length() > 0)
          if(mapa.containsKey(rec))
            mapa.put(rec, mapa.get(rec) + 1);
          else
            mapa.put(rec, 1);
      }
    }
    Set< String > kljucevi = mapa.keySet();
    int k = 1;
    for(String rec: kljucevi)
      System.out.printf("%-13s%3s%s", rec,
        mapa.get(rec), (k++ % 4 == 0)? "\n" : " | ");
  }
}
```

Primer sa TreeMap



Fajl

a	9	also	6	always	2	and	5	are	3
available	2	blocks	3	both	2	building	3	by	6
can	11	change	10	charts	3	choice	3	choose	3
choosing	3	command	2	commands	2	contained	2	controls	6
coordinate	6	cover	3	create	3	current	10	designed	3
diagrams	3	directly	6	document	17	easily	3	elements	3
footers	3	for	3	format	6	formatting	3	from	6
galleries	6	gallery	9	headers	3	home	6	in	7
include	3	insert	6	items	3	layout	3	lists	3
look	17	looks	2	most	3	new	3	of	14
offer	3	on	12	or	6	original	2	other	6
overall	6	page	3	pages	3	pictures	3	provide	2
quick	9	reset	2	restore	2	selected	6	set	2
so	2	specify	3	style	4	styles	5	tab	12
tables	3	template	2	text	12	that	8	the	56
theme	6	themes	2	these	3	they	3	to	13
use	5	using	9	when	3	with	6	you	17
your	13								

Ispis

Rad sa fajlovima i folderima. Klasa File

- Klasa **`File`** (paket `java.io`) pruža niz korisnih metoda u radu sa fajlovima i folderima (direktorijumima), od kojih je jedan broj dat u tabeli ispod.

Metoda	Opis
`exists`	Vraća `true` ako fajl ili folder predstavljen `File` objektom postoji, `false` u suprotnom.
`isFile`	Vraća `true` ako ime prosleđeno `File` konstruktoru odgovara fajlu, `false` u suprotnom.
`isDirectory`	Vraća `true` ako ime prosleđeno `File` konstruktoru odgovara folderu, `false` u suprotnom.
`isAbsolute`	Vraća `true` ako argumenti prosleđeni `File` konstruktoru odgovaraju apsolutnoj putanji do fajla ili foldera, `false` u suprotnom.
`getAbsolutePath`	Vraća `String` sa apsolutnom putanjom do fajla ili foldera.
`getName`	Vraća `String` sa imenom fajla ili foldera.
`getPath`	Vraća `String` sa putanjom do fajla ili foldera.
`getParent`	Vraća `String` sa roditeljskim (prvim iznad) folderom datog fajla ili foldera.
`length`	Vraća veličinu fajla, izraženu u bajtovima. Ako `File` objekat predstavlja folder, vraća se nedefinisana vrednost.
`lastModified`	Vraća vreme u `long` formatu (vrednost zavisi od platforme) poslednje modifikacije fajla ili foldera. Vraćena vrednost se može koristiti samo za poređenje sa drugim vrednostima koje vraća ova metoda.
`delete`	Briše predmetni fajl ili folder. U slučaju brisanja foldera, njegov sadržaj se mora izbrisati pre brisanja samog foldera. Vraća `true` ako je brisanje uspešno izvršeno i `false` u suprotnom.
`list`	Vraća niz `String` objekata koji predstavlja sadržaj foldera, odnosno `null` ako `File` objekat ne predstavlja folder.

Klasa File

- Koristeći objekte klase `File` ne možemo otvoriti fajlove, niti ih procesirati na bilo koji način. Ovi objekti se obično koriste u kombinaciji sa drugim `java.io` klasama da specificiraju sa kojim fajlovima i folderima se konkretno radi.
- `File` klasa ima četiri konstruktora.
- Pomenimo ovde samo konstruktor sa parametrom stringom koji određuje ime fajla ili foldera koje će biti pridruženo `File` objektu. Ime može sadržati i putanju do fajla/foldera. Putanja može uključiti samo neke ili sve foldere do željenog počev od korenog foldera (npr. `"C:\Prvi\Drugi\Treci"`).
- U nastavku dajemo primer korišćenja klase `File`. Od korisnika ćemo tražiti unos putanje do fajla ili foldera, a nakon toga, ukoliko predmetni fajl/folder postoji, odštampaćemo nekoliko opisnih poruka vezanih za fajl/folder. Dodatno, ako je u pitanju folder, odštampaćemo i njegov sadržaj.

Klasa File - Primer

```
import java.util.Scanner;
import java.io.File;

public class DemonstracijaKlaseFile {

   public static void main(String [] args) {
      Scanner unos = new Scanner(System.in);

      System.out.print("Uneti ime fajla ili foldera: ");
      File ime = new File(unos.nextLine());
      if(ime.exists()) {
         System.out.printf("%s\n%s\n%s\n%s\n%s\n%s",
                           ime.getName() + " postoji",
                           ime.isFile() ? "To je fajl" : "To nije fajl",
                           ime.isDirectory() ? "To je folder " : "To nije folder",
                           "Veličina: " + ime.length() + " bajta",
                           "Poslednja izmena: " + ime.lastModified(),
                           "Roditeljski folder: " + ime.getParent());
      }
      else
         System.out.printf("%s ne postoji", ime);

      if(ime.isDirectory()) {
         System.out.println("\n\nSadržaj foldera:");
         for(String f: ime.list())
            System.out.println(f);
      }

      unos.close();
   }
}
```

Klasa File - Primer

Izvršenje 1

```
Uneti ime fajla ili foldera: C:\Program Files (x86)\texstudio
texstudio postoji
To nije fajl
To je folder
Veličina: 4096 bajta
Poslednja izmena: 1618300520977
Roditeljski folder: C:\Program Files (x86)

Sadržaj foldera:
dictionaries
Doc.pdf
help
share
templates
texstudio.exe
TexTablet
translations
uninstall.exe
```

Izvršenje 2

```
Uneti ime fajla ili foldera: C:\Program Files (x86)\texstudio\Doc.pdf
Doc.pdf postoji
To je fajl
To nije folder
Veličina: 128946 bajta
Poslednja izmena: 1618234408258
Roditeljski folder: C:\Program Files (x86)\texstudio
```

Tekstualni fajlovi sa sekvencijalnim pristupom

- Demonstrirajmo rad sa tekstualnim fajlovima sa sekvencijalnim pristupom.
- Sekvencijalni pristup podrazumeva da se podaci upisuju i čitaju jedan za drugim. Nasuprot sekvencijalnom imamo slučajni pristup, gde bilo kad možemo pristupiti proizvoljnom podatku u toku.
- U prvom primeru ćemo kreirati tekstualni fajl na osnovu unosa korisnika.
- U drugom primeru ćemo otvoriti kreirani fajl i odštampati njegov sadržaj.
- Pomenute radnje ćemo demonstrirati pomoću novokreirane klase `Radnik`, prikazane na sledećem slajdu.

Klasa Radnik

`LocalDate` klasa predstavlja datume, bez vremenske zone, u ISO-8601 kalendarskom sistemu, npr. 2021-04-23.

Klasa `DateTimeFormatter` se koristi za formatiranje datuma. Metoda `ofPattern` ove klase kreira formater datuma na osnovu stringa šablona. Šablon je kombinacija predefinisanih slova i simbola za prikaz datuma. Na primer, šablon `"d MMM uuuu"` će formatirati datum `2021-04-23` kao `"23 Apr 2021"`.

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Radnik {
   private String ime;
   private String prezime;
   private LocalDate pocetak;
   private int sprema;
   private static final DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy");

   public Radnik() { this("", "", LocalDate.now(), 0); }
   public Radnik(String ime, String prezime, LocalDate pocetak, int sprema) {
          setIme(ime);  setPrezime(prezime);  setPocetak(pocetak);  setSprema(sprema); }

   public String getIme() { return ime; }
   public void setIme(String ime) { this.ime = ime; }
   public String getPrezime() { return prezime; }
   public void setPrezime(String prezime) { this.prezime = prezime; }
   public LocalDate getPocetak() { return pocetak; }
   public void setPocetak(LocalDate pocetak) { this.pocetak = pocetak; }
   public int getSprema() { return sprema; }
   public void setSprema(int sprema) { this.sprema = sprema; }

   @Override
   public String toString() {
      return String.format("%-10s %-12s sprema: %d  početak: %s",
                           this.getIme(), this.getPrezime(), this.getSprema(),
                           df.format(this.getPocetak()));
   }
}
```

Kreiranje txt fajla i upis u fajl

```
import java.util.*;
import java.io.*;
import java.time.LocalDate;

public class UpisRadnikaUFajl {

   private static Formatter izlaz;

   public void otvoriFajl() {
      try { izlaz = new Formatter("Radnici.txt", "UTF-8"); }
      catch (FileNotFoundException e) { System.err.println("Greška pri otvaranju."); System.exit(1); }
      catch (UnsupportedEncodingException e) { System.err.println("Kodiranje nije podržano.");
         System.exit(1); }
   }

   public void upisiRadnikeUFajl() {
      Radnik r = new Radnik();
      Scanner unos = new Scanner(System.in);

      System.out.print("Koliko radnika želite da unesete? ");
      int brojRadnika = unos.nextInt();
      System.out.println("Unesite radnike u formatu: <ime> <prezime> <datum> <sprema>");
      System.out.println("(<datum> unesite u formatu yyyy-mm-dd)");

      for(int i = 0; i < brojRadnika; i++) {
         r.setIme(unos.next());
         r.setPrezime(unos.next());
         r.setPocetak(LocalDate.parse(unos.next()));
         r.setSprema(unos.nextInt());
         izlaz.format("%s %s %s %d\n", r.getIme(), r.getPrezime(), r.getPocetak(), r.getSprema());
      }
      unos.close();
   }
```

```
   public void zatvoriFajl() {
      if(izlaz != null)
         izlaz.close();
   }
}
```

Kreiranje txt fajla i upis u fajl

```
public class TestiranjeUpisaUFajl {

   public static void main(String[] args) {
      UpisRadnikaUFajl fajlOper = new UpisRadnikaUFajl();
      fajlOper.otvoriFajl();
      fajlOper.upisiRadnikeUFajl();
      fajlOper.zatvoriFajl();
   }

}
```

Čitanje podataka iz txt fajla

```
import java.io.File;
import java.io.FileNotFoundException;
import java.time.LocalDate;
import java.util.Scanner;

public class CitanjeRadnikaIzFajla {

   private static Scanner ulaz;

   public static void main(String[] args) {
      otvoriFajl();
      citajRadnikeIzFajla();
      zatvoriFajl();
   }

   public static void otvoriFajl() {
      try { ulaz = new Scanner(new File("Radnici.txt")); }
      catch(FileNotFoundException e) { System.err.println("Greška pri otvaranju."); System.exit(1); }
   }

   public static void citajRadnikeIzFajla() {
      Radnik r = new Radnik();

      while(ulaz.hasNext()) {
         r.setIme(ulaz.next());
         r.setPrezime(ulaz.next());
         r.setPocetak(LocalDate.parse(ulaz.next()));
         r.setSprema(ulaz.nextInt());

         System.out.println(r);
      }
   }
```

```
   public void zatvoriFajl() {
      if(ulaz != null)
         ulaz.close();
   }
}
```