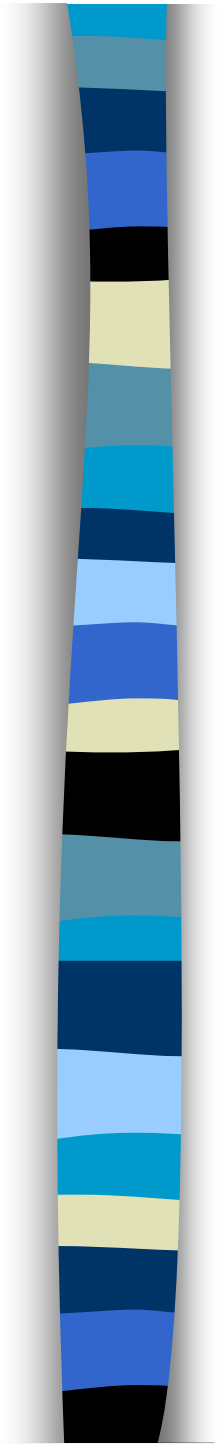


Java API

Java API

- Da bi se uspešno i efikasno programiralo u Javi, neophodno je poznavati i standardnu Java biblioteku - skup klasa i interfejsa koji se dobijaju zajedno sa prevodiocem, interpreterom i ostalim alatima.
- Ova biblioteka se još naziva i core **Java Application Programming Interface (Java API).**
- Java API znatno olakšava pravljanje aplikacija različitih vrsta, jer sadrži puno gotovih rešenja.
- Detaljan opis Java 8 API se može pronaći u zvaničnoj dokumentaciji.

Osnovne klase u Javi - paket java.lang

- Paket **java.lang** je jedini paket čijim se klasama i interfejsima može direktno pristupati u svakoj jedinici prevođenja, tj. bez prethodnog uvoza tih imena pomoću ključne reči import.
- Tipovi u ovom paketu su u tesnoj vezi sa pojedinim elementima programskog jezika Java.
- Neke od klasa i interfejsa u ovom paketu su:
- **Object** - nadklasa svih ostalih klasa (nema svoju nadklasu). Sadrži skup javnih metoda koji su zbog nasleđivanja vidljivi u svim ostalim klasama i nizovima.
- Neki od korisnijih metoda definisanih u ovoj klasi su **toString**, koji vraća tekstulani opis objekta, **equals** i **hashCode** za poređenje jednakosti objekata, i **wait**, **notify** i **notifyAll** za sinhronizaciju niti.

Osnovne klase u Javi - paket java.lang

- **Boolean, Character, Byte, Short, Integer, Long, Float, Double**
  - klase koje se koriste kada je proste tipove char, byte, short, int, long, float i double, respektivno, potrebno predstaviti objektima (na primer, u kolekcijama).
  - Pored toga, ove klase sadrže razne korisne konstante i metode koji se tiču prostih tipova i njihovih vrednosti.
    - Na primer, konstante **MAX_VALUE** i **MIN_VALUE** sadrže maksimalne, odnosno minimalne vrednosti celih i realnih brojeva i denisane su kao javna statička polja u odgovarajućim klasama.

Osnovne klase u Javi - paket java.lang

- **Math** sadrži mnoštvo statičkih metoda kojima su predstavljene razne matematičke funkcije.
  - Na primer, metod Math.abs vraća apsolutnu vrednost prosleđenog broja,
  - Math.sin računa sinus prosleđenog ugla (izraženog u radijanima), dok
  - Math.PI sadrži vrednost konstante π.
- **Throwable, Exception, RuntimeException** za rad sa izuzecima.
- Klasa **Thread** i interfejs **Runnable** omogućuju višenitno programiranje.
- **String** za rad sa stringovima, itd.

Korisne klase u paketu java.util

- Pored ranije obrađenih kolekcija, paket java.util sadrži klase i interfejse opšte namene, tj. klase i interfejse koje su od koristi u mnogim aplikacijama.
- Primeri su klase za rad sa datumom i vremenom, za manipulaciju nizovima, i za generisanje slučajnih brojeva.
- Slede kraći opisi i primeri upotrebe najinteresantnijih klasa i interfejsa.

Klase za rad sa datumom i vremenom

- Za rad sa datumom i vremenom se u Javi koristi više klasa:
  - Klasa **Date** čije instance predstavljaju datume i vremena.
  - Apstraktna klasa **Calendar** i njena podklasa **GregorianCalendar** čije instance takođe predstavljaju datume i vremena, ali omogućavaju i razna izračunavanja vezana za vreme. Na primer, upotrebom metoda ovih klasa moguće je dodavanje ili oduzimanje godina, meseci, dana, sati, itd. od nekog datuma.
  - Klasa **Locale** čije instance predstavljaju jezičke regione. Koristi se za postavljanje jezika i formata prilikom ispisa datuma i vremena, kao i naziva dana i meseci.
  - Klasa TimeZone čijim instancama se predstavljaju pojedine vremenske zone.

Klase `Locale` i `TimeZone` nije obavezno koristiti prilikom rada sa datumom i vremenom. Ako ih ne koristimo, aplikacija će odgovarajuća podešavanja preuzeti od operativnog sistema.

Klase za rad sa datumom i vremenom

- Rad sa datumom i vremenom u Javi može ponekad biti komplikovan.
- Na primer, često je potrebno kreirati objekat klase Date tako da predstavlja tačno određeni datum.
- Međutim, konstruktori koji prihvataju vrednosti za godinu, mesec, dan, itd. su označeni kao **deprecated**, što znači da ih ne treba koristiti (postoje samo zbog kompatibilnosti sa starim programima).
- Jedini dozvoljeni konstruktor je bez parametra, koji će datum i vreme postaviti na momenat kreiranja samog objekta.
- Za pravilnu konstrukciju objekta klase **Date** koji predstavlja tačno određeni datum i vreme se koristi klasa **Calendar**, kao što je prikazano u sledećem primeru.

Klase za rad sa datumom i vremenom

- Pravilna konstrukcija objekta klase Date tako da sadrži određeni datum

```
import java.util.Calendar;
import java.util.Date;

public class KreiranjeDatuma {
  public static void main(String[] args) {
    Calendar c = Calendar.getInstance();
    c.set(1982, 11, 28);
    Date d = c.getTime();
    System.out.println(d);
  }
}
```

- aplikacija transformiše datum u string koristeći podešavanja operativnog sistema. Na primer, ispis može izgledati ovako:

Tue Nov 28 17:04:27 CET 1982

- Dakle, 28. novembar 1982. godine je bio utorak. Pošto su metodu set prosleđeni samo godina, mesec i dan, kalendar i objekat klase Date sadrže vreme inicijalizacije kalendara po centralno-evropskom vremenu (CET).

Klase za rad sa datumom i vremenom

- Način ispisa je moguće promeniti.
- Klasa **DateFormat** iz paketa java.text služi za formatiranje ispisa datuma i vremena, dok se klasa Locale može iskoristiti za postavljanje jezika.
- Naredni primer ispisuje zadati datum na srpskom jeziku i u tačno određenom formatu:

Klase za rad sa datumom i vremenom

Primer 13.2: Postavljanje formata i jezika za ispis datuma

```
import java.text.DateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

public class DatumSrpski {
  public static void main(String[] args) {
    Locale jezik = new Locale("sr", "RS");
    DateFormat df = DateFormat.getDateTimeInstance(DateFormat.FULL,
        DateFormat.LONG, jezik);
    Calendar c = Calendar.getInstance();
    c.set(1982, 11, 28);
    Date d = c.getTime();
    String ispis = df.format(d);
    System.out.println(ispis);
  }
}
```

Poziv **df.format(d)** će prosleđeni datum formatirati koristeći ranije postavljena podešavanja. Rezultat izvršavanja ovog primera može biti:

Utorak, 28. novembar 1982. 17.22.25 CET

Klasa Arrays

Slično kao klasa Collections pomenuta ranije, Arrays sadrži statičke metode koji se mogu pogodno iskoristiti u radu sa nizovima. Pomoću metoda ove klase je moguće uraditi sledeće:

- Ispitati jednakost dva niza;
- Inicijalizovati sve ili samo neke elemente niza nekom vrednošću;
- Efikasno sortirati niz ili neki njegov deo;
- Efikasno pronaći element u sortiranom nizu i
- Pretvoriti niz u listu.

- Sledeći primer demonstrira upoređivanje, sortiranje i pretraživanje nizova pomoću klase Arrays.

Klasa Arrays

```
Nizovi nisu jednaki
Niz a: [-5, 0, 1, 2, 3, 4, 4, 6, 9, 13]
Niz b: [33, -3, -4, 4, 8, 29, 1, 2, 7, 1]
Element 6 se u nizu nalazi na poziciji 7
```

```
import java.util.Arrays;

public class KlasaArrays {
  public static void main(String[] args) {
    int[] a = {3, 1, 4, 2, 6, 9, 13, -5, 4, 0};
    int[] b = {33, -3, 4, 29, 8, -4, 1, 2, 7, 1};

    // poredjenje nizova
    if (Arrays.equals(a, b))
      System.out.println("Nizovi su jednaki");
    else
      System.out.println("Nizovi nisu jednaki");

    // sortiranje
    Arrays.sort(a);
    // sortira se samo segment od elementa sa indeksom 2 (ukljucujuci)
    // do elementa sa indeksom 6 (iskljucujuci)
    Arrays.sort(b, 2, 6);

    // stampanje nizova
    System.out.println("Niz a: " + Arrays.toString(a));
    System.out.println("Niz b: " + Arrays.toString(b));

    // binarno pretrazivanje niza
    int indeks = Arrays.binarySearch(a, 6);

    System.out.println("Element 6 se u nizu nalazi na poziciji " +
        indeks);
  }
}
```

Klasa Arrays

- Kod provere jednakosti nizova se ispituje da li nizovi imaju jednak broj elemenata i da li su odgovarajući elementi jednaki.
- Metod sort služi za sortiranje elemenata niza, dok metod binarySearch pronalazi zadati element u sortiranom nizu.
- Na primeru je prikazan rad sa nizovima čiji su elementi tipa int.
- Klasa Arrays sadrži slične metode za sve ostale tipove elemenata niza, uključujući i referencijalne tipove.
- Ako je tip elementa niza neki referencijalni tip, onda se sortiranje i binarno traženje takođe može realizovati, ali se prethodno mora definisati način upoređivanja objekata koji se nalaze u nizu, slično kao kod kolekcija.

Najefikasnije kopiranje nizova omogućuje klasa `java.lang.System` svojim statičkim metodom `arraycopy`.

Klasa Random

- Klasa Random se koristi za generisanje pseudoslučajnih brojeva. Sledeći primer demonstrira njeno korišćenje.

```
import java.util.Random;
import static java.lang.System.out;

public class KlasaRandom {
  public static void main(String[] args) {
    Random sluc = new Random();
    out.println(sluc.nextBoolean());   // slucajna logicka vrednost
    out.println(sluc.nextDouble());    // slucaj double iz [0.0, 1.0)
    out.println(sluc.nextInt());       // slucajan int
    out.println(sluc.nextInt(10));     // slucajan int manji od 10
    out.println(sluc.nextGaussian()); // slucajan double sa
          // normalnom raspodelom slucajne promenljive (m=0.0, d=1.0)
  }
}
```

Ulazno-izlazne operacije i paket java.io

- Paket **java.io** sadrži tipove koji se koriste kod različitih ulazno-izlaznih operacija u Java aplikacijama.
- Klase iz ovog paketa se koriste kada je potrebno čitati iz ili pisati u datoteke, čitati sa tastature, koristiti mrežnu komunikaciju i tome slično.
- Rad sa tipovima iz paketa java.io je jednostavan bez obzira na to o kojoj vrsti ulazno-izlazne operacije se radi.
- Štaviše, često se koriste iste klase i to na isti način bez obzira na to gde se upisuju podaci ili odakle se oni čitaju.
- Razlika je najčešće samo u pozivu konstruktora, jer se tu navodi cilj odnosno izvor podataka.

Klase InputStream i OutputStream

- Mnoge klase u paketu java.io su podklase apstraktnih klasa **InputStream** i **OutputStream**.
- InputStream je namenjen ulaznim operacijama (čitanju), dok je OutputStream namenjen izlaznim operacijama (pisanju).
- Obe klase sadrže metode koje rade na nivou bajtova, što je najčešće suviše nizak nivo apstrakcije za potrebe prosečnog programa.
- Zbog toga postoje druge klase koje su podklase ovih klasa i koje omogućavaju čitanje i pisanje na višem nivou apstrakcije.

Klase Reader i Writer

- Apstraktne klase **Reader** i **Writer** imaju sličnu ulogu kao i InputStream i OutputStream.
- Razlika je u tome što su Reader i Writer specijalno namenjene radu sa tekstualnim ulazno-izlaznim operacijama, dok se InputStream i OutputStream (tj. njihove podklase) mogu koristiti za sve vrste ulazno-izlaznih operacija.
- Klase Reader i Writer imaju nekoliko podklasa u paketu java.io.

Rad sa fajlovima

- Osnovne operacije operativnog sistema za rad sa fajlovima je moguće izvršiti pomoću klase **File**.
- Instancama ove klase se predstavljaju fajlovi i direktorijumi.
- Između ostalog, metodima ove klase je moguće uraditi sledeće:
- proveriti da li dati fajl uopšte postoji, proveriti da li je dozvoljeno čitanje tog fajla, da li je dozvoljeno upisivanje novih podataka u fajl, proveriti kada je fajl poslednji put modikovan, promeniti osobine fajla, preimenovati fajl, obrisati fajl, kreirati novi prazan fajl, kreirati novi direktorijum, izlistati sadržaj direktorijuma, i td.
- Sledeći primer prikazuje neke od nabrojanih operacija.

Rad sa fajlovima

```
import java.io.File;

public class OsobineFajla {
  public static void main(String[] args) {
    File f = new File(args[0]); // prvi argument je puno ime fajla
    if (f.exists()) {

      System.out.println("Fajl " + args[0] + " postoji.");
      if (f.isDirectory()) {
        System.out.println("U pitanju je direktorijum. " +
          "Njegov sadrzaj je:");
        File[] sadrzaj = f.listFiles();
        for (File sub : sadrzaj)
          System.out.println(sub.getName());
      }
      else {
        System.out.println("U pitanju je obican fajl.");
        System.out.println("Njegova velicina je " + f.length() +
          " bajtova.");
      }
    }
    else
      System.out.println("Fajl " + args[0] + " ne postoji.");
  }
}
```

Rad sa fajlovima

- Prilikom čitanja iz i pisanja u fajlove, najčešće se koriste sledeće klase:
  FileInputStream, FileOutputStream, FileReader i FileWriter.
- Prve dve klase služe za rad i sa binarnim i sa tekstualnim fajlovima (pre svega binarnim), dok druge dve služe za rad samo sa tekstualnim fajlovima.
- Sledeći primer demonstrira čitanje iz fajla upotrebom klase **FileReader**, kao i upisivanje u fajl upotrebom klase **FileWriter**.
- Međutim, obe klase rade na relativno niskom nivou, i omogućavaju samo čitanje i pisanje pojedinačnih karaktera.
- Zbog toga ćemo iskoristiti **BufferedReader** i **PrintWriter** za čitanje i pisanje stringova.

Rad sa fajlovima

Primer 13.6: Pisanje u fajl i čitanje iz fajla

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

public class UlazIzlazFajl {
  // ispisuje niz stringova str u fajl f
  public static void pisi(String[] str, File f) {
    try (PrintWriter izlaz = new PrintWriter( new FileWriter(f) )) {
      for (String s : str)
        izlaz.println(s);  //upis u fajl
    } catch (IOException e) {
      System.err.println( e.getMessage() );
    }
  }

  // ucitava stringove iz fajla f u kolekciju
  public static List<String> citaj(File f) {
    List<String> list = new ArrayList<>();
    try (BufferedReader ulaz = new BufferedReader(new FileReader(f))) {
      String s = ulaz.readLine();  // citanje iz fajla
      while (s != null) {
        list.add(s);
        s = ulaz.readLine();
```

Rad sa fajlovima

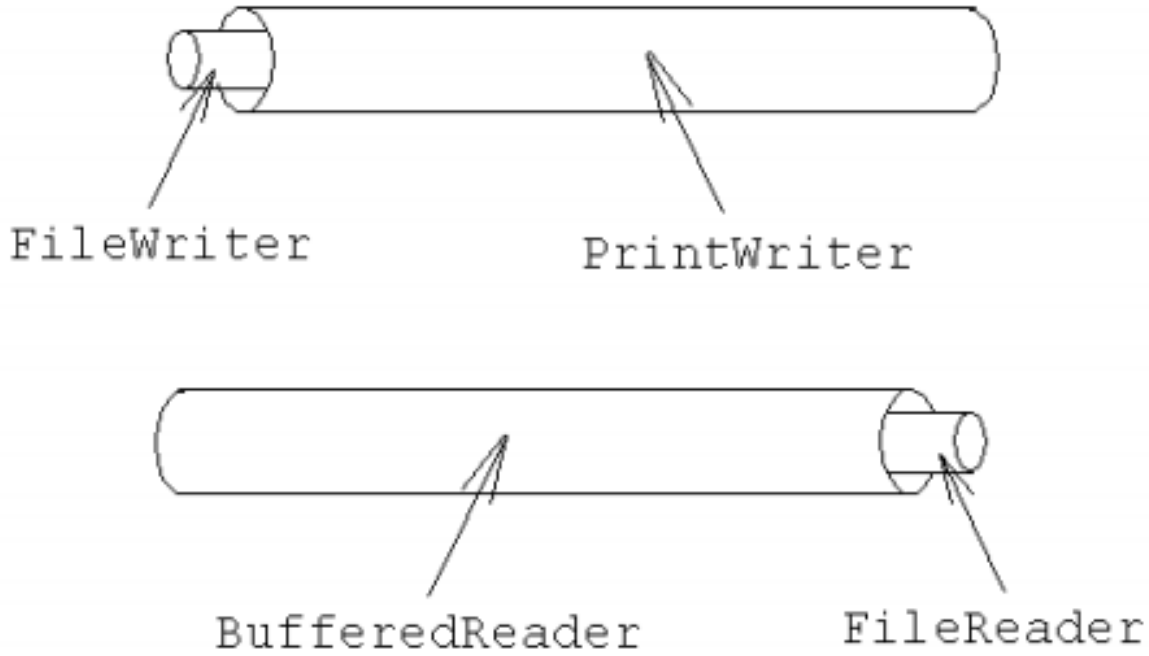
```
            }
        } catch (IOException e) {
            System.err.println( e.getMessage() );
        }
        return list;
    }

    public static void main(String[] args) {
        File f = new File("a.txt");
        f.deleteOnExit(); // obrisi fajl pri izlasku
        pisi(args, f);
        List<String> procitano = citaj(f);
        System.out.println("Sadrzaj fajla je:");
        for (String s : procitano)
            System.out.println(s);
    }
}
```

Rad sa fajlovima

- Konstruktori klasa **FileWriter** i **FileReader** prihvataju putanju do fajla u obliku stringa, ili objekat klase File. U datom primeru je korišćen drugi, ovde pogodniji pristup, jer metodi čitaju i upisuju u isti fajl.
- Takođe, poziv **f.deleteOnExit()** osigurava da će fajl biti obrisan kada program (tj. virtuelna mašina) završi sa radom.
- Način na koji smo u primeru iskoristili klase PrintWriter i BufferedReader je karakterističan za rad sa paketom java.io.
- Prilikom kreiranja klase PrintWriter u pozivu konstruktora smo kao parametar prosledili instancu klase FileWriter.
- Tako dobijen objekat PrintWriter omogućuje rad sa fajlom na višem nivou (upis stringova u fajl), pri čemu on interno upisuje pojedinačne karaktere koristeći dati FileWriter.
-

Rad sa fajlovima

- Na sličan način funkcionišu i klase BufferedReader i FileReader koje se koriste u metodu citaj.



Ulazno-izlazne operacije nad vrednostima prostih tipova podataka

- Ulazno-izlazne operacije nad vrednostima prostih tipova podataka se najefikasnije mogu realizovati pomoću klasa **DataInputStream** i **DataOutputStream**.
- Sledeći primer demonstrira njihovu upotrebu. Vrednosti se najpre upisuju u fajl, a posle se iz njega čitaju.
- Pošto se radi o binarnom, a ne tekstualnom fajlu, koriste se klase **FileInputStream** i **FileOutputStream**.

Klase DataInputStream i DataOutputStream

```
import java.io.*;

public class DataInOut {
  public static void upisi(File f) {
    try (DataOutputStream izlaz = new DataOutputStream(
        new FileOutputStream(f))) {
      izlaz.writeBoolean( true );
      izlaz.writeDouble( 3.14 );
      izlaz.writeInt( -9 );
      izlaz.writeShort( 2 );
    } catch (IOException e) {
      System.err.println( e.getMessage() );
    }
  }

  public static void procitaj(File f) {
    try (DataInputStream ulaz = new DataInputStream(
        new FileInputStream(f))) {
      System.out.println("Iz fajla su procitane sledece vrednosti:");
      System.out.println( ulaz.readBoolean() );
      System.out.println( ulaz.readDouble() );
      System.out.println( ulaz.readInt() );
      System.out.println( ulaz.readShort() );
    } catch (IOException e) {
      System.err.println( e.getMessage() );
    }
  }
  public static void main(String[] args) {
    File f = new File("brojevi.dat");

    f.deleteOnExit();
    upisi(f);
    procitaj(f);
  }
}
```

```
Iz fajla su procitane sledece vrednosti:
true
3.14
-9
2
```

Ulazno-izlazne informacije nad objektima

- Nad objektima se mogu vršiti ulazno-izlazne operacije.
- Na primer, objekti se mogu zapisati u fajl, a posle se iz njega mogu pročitati.
- Time se dobija perzistentnost objekata: očuvanje stanja i nakon prekida rada programa.
- Uslov za ovu funkcionalnost je da klasa objekta implementira interfejs **java.io.Serializable**, kao i da tipovi svih njenih nestatičkih polja koja nisu označena modifikatorom transient takođe implementiraju ovaj interfejs (isto važi i za njihova polja, itd.).
- Interfejs Serializable nema ni jedan metod i služi samo kao marker. Njega implementiraju sve najčešće korišćene klase iz Java API, kao i svi nizovi.

Ulazno-izlazne informacije nad objektima

- Za ulazno-izlazne operacije nad **objektima** se koriste klase **ObjectInputStream** i **ObjectOutputStream**.
- Pored toga što omogućuju ulazno-izlazne operacije nad objektima, ove klase omogućavaju i ulazno-izlazne operacije nad vrednostima prostih tipova podataka.
- To znači da se ove klase mogu koristiti umesto ranije opisanih klasa DataInputStream i DataOutputStream.
- Klase koje implementiraju interfejs java.io.Serializable bi trebalo da definišu i vrednost polja **serialVersionUID**. Ovo polje označava serijski broj, odnosno verziju klase.
- Prilikom učitavanja objekta iz fajla, Java će uporediti verziju klase u fajlu sa verzijom u programu. Ukoliko je, na primer, objekat upisan u fajl jako davno, a klasa je u međuvremenu menjana i ima novi serijski broj, biće generisan izuzetak.

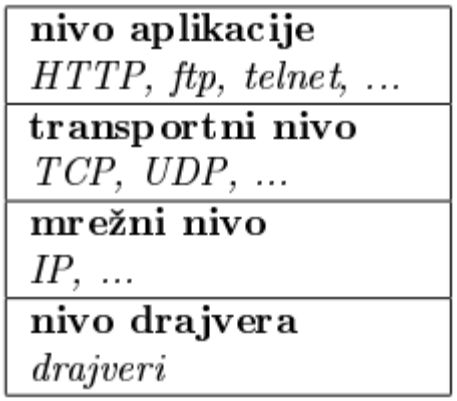
Ulazno-izlazne informacije nad objektima

- Objekat se u fajl upisuje tako što se najpre pretvori u niz bajtova koji se onda upisuju u fajl.
- Ovaj postupak pretvaranja objekta u niz bajtova se naziva **serijalizacija**.
- Kao što je rečeno, serializacijom se u niz bajtova pretvaraju sva polja objekta, izuzev statičkih polja i polja koja su označena modikatorom transient.
- Obrnuti postupak, rekonstrukcija objekta iz niza bajtova, naziva se **deserijalizacija**.

Način na koji se instance pojedinih klasa upisuju u fajl se može odrediti eksplicitno ako se umesto interfejsa `java.io.Serializable` koristi interfejs `java.io.Externalizable`.

Mrežno programiranje - paket java.net

- Mrežna aplikacija se sastoji od dva ili više delova koji se izvršavaju na različitim računarima spojenim mrežom.
- Kod realizacije mrežne komunikacije možemo uočiti više nivoa.



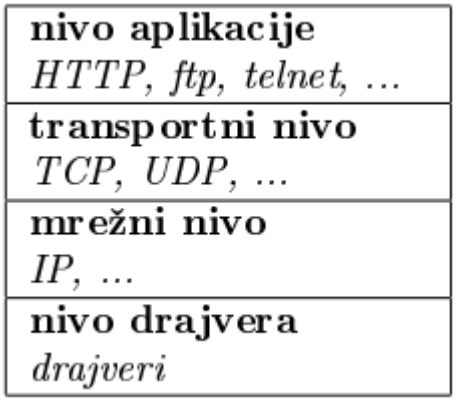
- Na najvišem nivou se nalazi nivo aplikacije. Tu se nalaze protokoli koje koriste pojedine aplikacije kada međusobno razmenjuju podatke.
- Najpoznatiji takav protokol je HTTP (Hypertext Transfer Protocol) koji koriste Internet pretraživači za komunikaciju sa WWW serverima.

Mrežno programiranje - paket java.net

- Nivo aplikacije koristi usluge nižeg nivoa, tj. transportnog nivoa.
- Transportni nivo se sastoji od protokola koji omogućuju transport podataka sa jednog računara na drugi. Dva najpoznatija transportna protokola su TCP (Transport Control Protocol) i UDP (User Datagram Protocol).
- Sam transportni nivo koristi usluge još nižeg, tzv. mrežnog nivoa. Ovom nivou pripada IP (Internet Protocol) koji zajedno sa TCP-em predstavlja osnovu moderne internet komunikacije.
- Protokoli mrežnog nivoa upravljaju pojedinim hardverskim komponentama koristeći njihove drajvere.

TCP i UDP

- Programiranje mrežnih aplikacija u Javi je veoma jednostavno, za razliku od mrežnog programiranja u drugim jezicima.
- Paket java.net sadrži tipove koji omogućuju rad sa mrežnim aplikacijama na najvišem, aplikacionom nivou.
- Ipak, da bi se klase i interfejsi iz ovog paketa pravilno koristili, potrebno je osnovno poznavanje transportnih protokola TCP i UDP.



Nivoi mrežne komunikacije

TCP i UDP

- **TCP** se koristi kada je potrebno uspostaviti stabilan i pouzdan komunikacioni kanal između dva računara.
- Podaci koji se šalju na ovaj način će sigurno stići do odredišta ako se veza ne prekine, i to u redosledu u kojem su poslati.
- Prednost TCP-a je, tako, njegova pouzdanost.
- Nedostatak je u većim zahtevima za resursima, jer svi računari kroz koje prolazi komunikacioni kanal održavaju uspostavljenu vezu sve dok se komunikacija ne završi.
- HTTP, ftp i telnet protokoli se koriste sa TCP transportnim protokolom.

TCP i UDP

- **UDP** transportni protokol pruža manju pouzdanost u mrežnoj komunikaciji.
- Podaci poslati ovim protokolom ne moraju uvek stići do odredišta, a redosled poslatih i primljenih podataka između dva ista računara može biti različit.
- UDP se koristi za slanje i primanje tzv. **datagrama**. Datagram se sastoji od niza bajtova kojima su predstavljeni podaci, kao i od adrese odredišta.
- Nakon što pošalje datagram, pošiljalac ne dobija nikakvu povratnu informaciju o tome da li je datagram stigao do odredišta ili ne.
- Datagram se kreće od jednog računara u mreži do drugog, sve dok ne stigne do cilja.
- Nakon što primi i prosledi datagram, usputni računar u mreži ga „zaboravlja“.

TCP i UDP

- Prednost UDP-a je u njegovim malim zahtevima za resursima, jer se ne uspostavlja nikakav komunikacioni kanal.
- Nedostatak UDP komunikacije je njena nepouzdanost.
- Ipak, za neke aplikacije je UDP idealan izbor.
- Na primer, ako jedan program datagramima svakih nekoliko sekundi šalje drugom programu trenutnu temperaturu vazduha, onda neće biti velika šteta ako se neki zagube, ili ako dva datagrama stignu u drugačijem redosledu od onog kojim su poslati.

Portovi računara

- Bez obzira na to da li se koristi TCP ili UDP transportni protokol, za uspostavljanje mrežne komunikacije između dva programa nije dovoljno navesti samo adresu ciljnog računara.
- Neophodno je navesti i port ciljnog računara na kome odgovarajući program očekuje podatke.
- Svaki računar najčešće ima samo jednu fizičku vezu sa preostalim računarima u mreži.
- Pošto se na računaru istovremeno može izvršavati više programa koji koriste mrežnu komunikaciju, svakom od tih programa se pridružuje po jedan broj, tj. port.

Portovi računara

- Dakle, pomoću porta se određuje kom programu su namenjeni podaci pristigli preko mreže.
- Broj porta može biti bilo koji broj od 0 do 65535.
- Međutim, brojevi od 0 do 1023 su rezervisani za poznate servise i njih ne bi trebalo koristiti u druge svrhe.
- Tako je, na primer, standardni port za HTTP port 80.

Klasa URL

- URL je skraćenica od Uniform Resource Locator.
- Pomoću URL-a se na standardni način označavaju resursi na lokalnom ili, što je češći slučaj, na udaljenom računaru.
- Resurs na koji URL pokazuje je najčešće neki fajl, ali može biti i dinamički sadržaj, tj. sadržaj koji se generiše prilikom pristupa resursu.
- URL se sastoji od nekoliko delova. Primer jednog URL-a je:
- http://www.dmi.uns.ac.rs/files/14/m_05_ai4.pdf

Klasa URL

- URL se sastoji od nekoliko delova. Primer jednog URL-a je:

http://www.dmi.uns.ac.rs/files/14/m_05_ai4.pdf

- Ovaj URL označava fajl na udaljenom računaru, i sastoji se iz tri dela:
  - Oznaku protokola kojim se resursu pristupa (http);
  - Ime računara na kom se resurs nalazi (www.dmi.uns.ac.rs); i
  - Putanju do i ime resursa na tom računaru (files/14/m_05_ai4.pdf).

Klasa URL

- Osim navedenih, URL može sadržati još podataka, poput korisničkog imena i lozinke, i reference na resurs.
- Puna struktura URL-a izgleda ovako:

protokol://ime:lozinka@racunar:port/putanja?parametri#referenca

- Instancama klase URL se u Javi predstavljaju URL-ovi.
- Metodi ove klase omogućuju dobijanje podataka o pojedinim delovima URL-a.
- Sledeći primer koristi instancu klase URL kako bi odštampao sadržaj HTML stranice.

Klasa URL

```
import java.net.*;
import java.io.*;

public class KlasaURL {
  public static void main(String[] args) throws Exception {
    URL nastava = new URL("http://www.dmi.uns.ac.rs/studije");
    try (BufferedReader ulaz = new BufferedReader(
        new InputStreamReader(nastava.openStream()))) {
      String red = ulaz.readLine(); // citamo prvi red
      while (red != null) {
        System.out.println(red);
        red = ulaz.readLine(); // citamo sledeci red
      }
    }
  }
}
```

- Klasa URL omogućuje mrežno programiranje na veoma visokom nivou apstrakcije.
- Za pisanje programa iz primera nije potrebno znati bilo šta o pojedinim protokolima; jednostavno smo naveli adresu resursa i onda smo ga pročitali.
- Pošto se resursu pristupa HTTP protokolom (tako je navedeno u URL-u), napravljena instanca klase URL koristi HTTP protokol

Klasa Socket

- Kod TCP mrežne komunikacije se između dva računara uspostavlja komunikacioni kanal.
- Krajevi tog kanala se nazivaju soketima (eng. socket). Komunikacija pomoću TCP-a se najčešće koristi kod klijent-server aplikacija.
- Komunikacija između klijenta i servera ima uvek istu formu.
  - Klijent najpre uspostavi komunikacioni kanal sa serverom.
  - Nakon toga, obično klijent šalje neki zahtev ili upit serveru, a
  - server mu odgovara.

Klasa Socket

- Tipičan primer klijent-server odnosa imamo kod internet pretraživača i WWW servera.
- Internet pretraživač je klijent zato što on inicira komunikaciju sa WWW serverom i šalje mu zahtev za dobijanje sadržaja neke stranice.
- WWW server, na primer www.dmi.uns.ac.rs, je server zato što on čeka na iniciranje komunikacije od strane klijenta, nakon čega odgovara na njegov zahtev i šalje mu traženu stranicu.
- Instancama klase java.net.Socket se predstavljaju soketi koji se koriste kod TCP komunikacije.

Klasa ServerSocket

- Osim za programiranje klijenata, paket java.net se takođe koristi i za programiranje servera u TCP komunikaciji.
- U tu svrhu se koristi klasa **ServerSocket**.
- Server je program koji čeka na nekom portu da mu se klijent javi.
- Nakon što stigne zahtev od klijenta, uzima se prvi slobodni port na računaru servera, i komunikacija sa klijentom se odvija preko njega.
- O svemu ovome vodi računa instanca klase ServerSocket.
- Jedan server često opslužuje više klijenata koji mu istovremeno pristupaju.
- Zbog toga komunikaciju sa svakim klijentom treba izvršavati u posebnoj niti programa. Na ovaj način se svi klijenti istovremeno opslužuju, bez čekanja.

Distribuirane aplikacije i paket java.rmi

- Paket **java.rmi** sadrži klase i interfejse koji omogućuju pozivanje metoda prostorno udaljenih objekata.
- Poziv metoda se, zajedno sa vrednostima parametara, transportuje do ciljnog Java objekta preko računarske mreže.
- Nakon izvršenja metoda se eventualna povratna vrednost vraća mrežom nazad.
- Na ovaj način se udaljeni objekti mogu koristiti na sličan način kao i lokalni objekti.
- Time paket java.rmi bitno pojednostavljuje pravljenje distribuiranih mrežnih aplikacija.
- RMI je akronim od **Remote Method Invocation**, što u prevodu znači udaljeno pozivanje metoda.

Distribuirane aplikacije i paket java.rmi

- Iako dosta slično, korišćenje udaljenih objekata malo razlikuje od korišćenja lokalnih objekata:
  - Da bi se metodi nekog objekta mogli pozivati preko mreže, mora postojati interfejs koji nasleđuje **java.rmi.Remote**. Objekat dostupan preko mreže implementira ovaj interfejs. Pri tome, samo se metodi koji su navedeni u interfejsu mogu svi pozivati preko mreže.
  - Svi metodi koji se pozivaju udaljeno moraju u svom zaglavlju deklarisati da bacaju izuzetak tipa **java.rmi.RemoteException**. Ovaj izuzetak će biti generisan ako nastanu neki problemi prilikom udaljenog pozivanja metoda.
  - Tipovi stvarnih parametara metoda i stvaran tip rezultata metoda (ako se ne radi o void metodu) moraju biti ili prosti tipovi ili klase koje implementiraju interfejs **java.io.Serializable**. Kao što je ranije rečeno, ovaj interfejs se koristi prilikom serijalizacije i deserijalizacije objekata, odnosno prilikom transporta parametara i povratne vrednosti udaljenog metoda preko mreže.

Distribuirane aplikacije i paket java.rmi

- Pomoću paketa **org.omg.CORBA** se takođe prave distribuirane aplikacije gde Java objekti, koristeći standardni CORBA protokol, pozivaju metode udaljenih objekata, i udaljeni objekti pozivaju njihove metode.
- Pošto je CORBA protokol nezavisan od programskog jezika koji se koristi, na ovaj način je moguće uspostaviti međusobnu mrežnu komunikaciju između objekata kreiranih u Javi, C++, Delphi-ju i ostalim jezicima.
- Za razliku od CORBA protokola, RMI protokol se koristi samo za Java objekte.

Teorijske vežbe

Objektno-orjentisano programiranje

Zadatak 9

- Napisati Java aplikaciju koja u prozoru dimenzija 400x400 piksela simulira kretanje kraljice po šahovskoj tabli.
- Kraljica se inicijalno nalazi u gornjem-levom uglu.
- Sva polja koja kraljica napada su označena tekstom "X".
- Kada korisnik klikne levim tasterom miša na neko polje, kraljica se pomera na to polje ukoliko je potez dozvoljen, odnosno aplikacija prikazuje odgovarajuću poruku o grešci.
- Klik bilo kojim drugim tasterom miša vraća kraljicu na početnu poziciju.

Zadatak 9

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

@SuppressWarnings("serial")
public class Kraljice extends JFrame {
    private final static int DIM = 8;
    private JLabel[][] labele;
    private Icon kraljica;
    private int red, kol;
```

```
public Kraljice() {
    setLayout(new GridLayout(DIM, DIM));

    labele = new JLabel[DIM][DIM];
    for (int i = 0; i < DIM; i++)
        for (int j = 0; j < DIM; j++) {
            labele[i][j] = new JLabel();
            labele[i][j].setOpaque(true);
            if ((i + j) % 2 == 0)
                labele[i][j].setBackground(Color.WHITE);
            else
                labele[i][j].setBackground(Color.GRAY);
            labele[i][j].setHorizontalAlignment(JLabel.CENTER);

            final int red = i, col = j;
            labele[i][j].addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                    if (e.getButton() == MouseEvent.BUTTON1) {
                        if (napadnuto(red, col))
                            postaviKraljicu(red, col);
                        else
                            JOptionPane.showMessageDialog(Kraljice.this,
                                    "Kraljica ne napada to polje.");
                    } else
                        postaviKraljicu(0, 0);
                }
            });

            add(labele[i][j]);
        }
```

Zadatak 9

```
        kraljica = new ImageIcon("kraljica.png");
        postaviKraljicu(0, 0);
    }

    private void postaviKraljicu(int i, int j) {
        labele[red][kol].setIcon(null);
        red = i;
        kol = j;
        labele[red][kol].setIcon(kraljica);
        oznaciNapadnutaPolja();
    }
```

Zadatak 9

```
private void oznaciNapadnutaPolja() {
    for (int i = 0; i < DIM; i++)
        for (int j = 0; j < DIM; j++) {
            if (i == red && j == kol)
                labele[i][j].setText(""); // ovde je kraljica
            else
                labele[i][j].setText(napadnuto(i, j) ? "X" : "");
        }
}

private boolean napadnuto(int i, int j) {
    if (i == red || j == kol) // vrsta/kolona
        return true;
    if (Math.abs(i - red) == Math.abs(j - kol)) // dijagonala
        return true;
    return false;
}
```

Zadatak 9

```
    public static void main(String[] args) {
        Kraljice k = new Kraljice();
        k.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        k.setSize(400, 400);
        k.setTitle("Kraljice");
        k.setVisible(true);
    }
}
```