



# 12

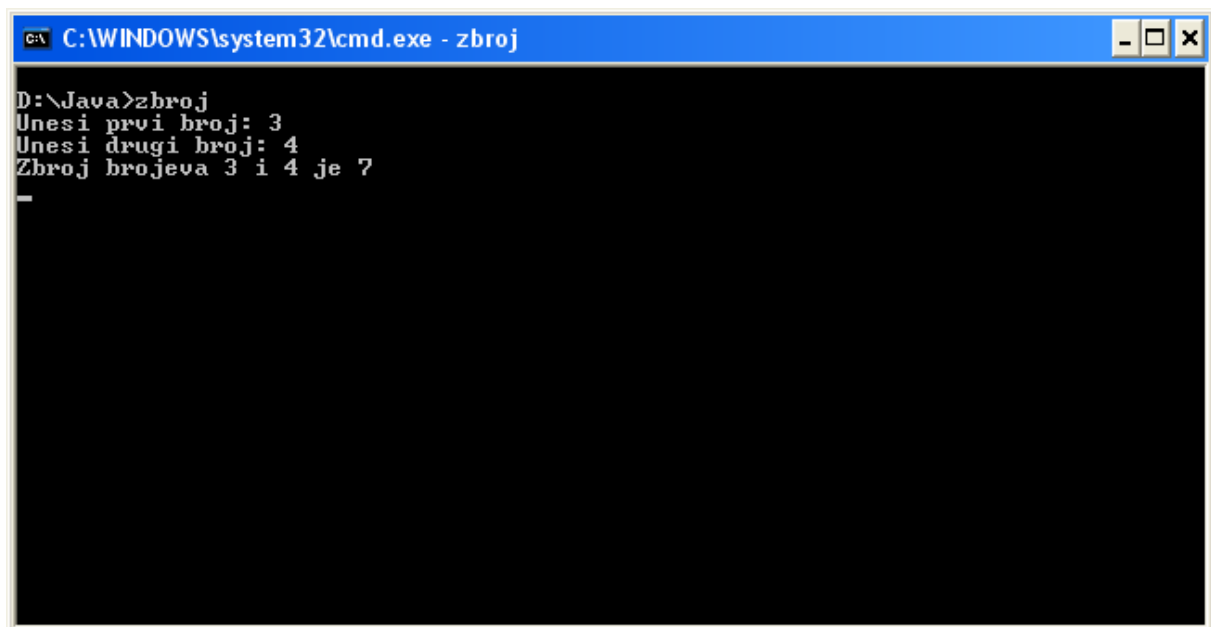
## Unos i ispis podataka

- Streamovi
- Standardni ulaz i izlaz
- Tekstualne datoteke

## Streamovi

Jedna od osnovnih pretpostavki programa je da na osnovu ulaznih podataka u konačnom broju koraka daju ispravne izlazne rezultate. Unositi podatke u program te prikazivati rezultate moguće je na različite načine. Mi smo u početku podatke unosili prosljeđivanjem vrijednosti parametara metodama kroz BlueJ razvojno okruženje. U posljednjem smo poglavlju naučili kako kreirati grafičko korisničko sučelje u Javi, te smo programima vrijednosti prosljeđivali kroz elemente grafičkog korisničkog sučelja (najčešće okvire za tekst). Rezultate obrade smo isto tako prikazivali kroz elemente BlueJ razvojnog okruženja ili na nekom elementu grafičkog korisničkog sučelja. U praksi je unos podataka i prikaz rezultata obrade preko elemenata grafičkog korisničkog sučelja uistinu vrlo čest.

Ne tako davno, dok još nisu postojali moderni operativni sustavi s grafičkim korisničkim sučeljima programi su se pokretali iz tzv. komandnog prompta. Na isti su se način prosljeđivali i ulazni podaci takvim programima (*slika 12 – 1*). Rezultati su se također ispisivali u komandnoj liniji. Takve programe možemo pisati i danas i zovemo ih **konzolnim programima**.



**Slika 12 – 1:** Program s tekstualnim korisničkim sučeljem

Međutim isto tako se nerijetko podaci, koji su predmet obrade uzimaju iz prethodno definiranih dokumenata ili tablica baza podataka. Rezultati obrade se isto tako mogu pohranjivati u datoteke ili baze podataka i kao takvi mogu poslužiti kao ulazni podaci za neke druge programe,...

Podatke je isto tako moguće dobivati iz nekog specijalnog mjernog uređaja (termometar,...) s drugih računala koristeći računalne mreže,...

U nastavku ćemo se baviti konzolnim programima koji s podacima manipuliraju pomoću komandnog prompta, odnosno programima koji podatke unose iz datoteka ili rezultate obrade spremaju u datoteke. Nešto kasnije ćemo dati i primjere kako razmjenjivati podatke s drugim računalima putem računalne mreže te kako komunicirati s bazama podataka.

Za unos odnosno ispis podataka najčešće se koriste **streamovi**. **Stream** predstavlja bilo koji izvor s kojeg možemo uzimati ili na koji možemo stavljati podatke. U Javi postoje dva osnovna tipa streamova streama:

- bajtovni stream
- znakovni stream

## Bajtovni streamovi

Bajtovni streamovi nam omogućavaju jednostavan unos odnosno ispis bajtova. Rad s bajtovnim streamovima omogućavaju nam dvije osnovne klase:

- **InputStream** – za unos bajtova
- **OutputStream** – za ispis bajtova

Klasa **InputStream** i sve klase koje ju nasljeđuju ima metodu **read ()** koja učitava sljedeći bajt s ulaza. S obzirom da **read ()** uzima bajt s ulaza, ova metoda vraća tip podataka **integer**.

Isto tako klasa **OutputStream** i sve klase koje ju nasljeđuju ima metodu **write ()** za ispis jednog bajta.

## Znakovni streamovi

Znakovne streamove koristimo za unos i ispis znakova. Za rad s znakovnim streamovima definirane su dvije klase:

- **Reader**
- **Writer**

Osim ovih dviju osnovnih klasa definirano je još i niz drugih klasa koje ih nasljeđuje. Klasa **Reader** i sve klase koje ju nasljeđuju imaju metodu **read ()** za unos jednog znaka. Isto tako klasa **Writer** i sve klase koje ju nasljeđuju imaju metodu **write ()** za ispis jednog znaka.

Sve navedene klase za rad sa streamovima nalaze se u paketu **java.io.\***, te se moraju uključiti u program naredbom **import**.

---

## Standardni ulaz i izlaz

Svi programi u Javi inicijalno imaju uključen paket **System**. Unutar tog paketa između ostalog nalaze se tri stream varijable koje su nam trenutno bitne. Radi se o svojstvima: **in**, **out**, **err**. Sva tri svojstva definirana su kao *public* i *static*, što znači da im možemo pristupiti direktno bez kreiranja objekta tipa **System**.

**System.in** se odnosi na standardni ulaz koji je u principu tipkovnica. **System.out** je standardni izlazni stream, inicijalno je to konzola (komandna linija). **System.err** je standardni stream za rad s greškama, inicijalno je to i ovdje konzola.

**System.in** je tipa **InputStream**, dok su **System.out** i **System.err** tipa **PrintStream**. Kao što možemo primijetiti radi se o bajtovnim streamovima koji se najčešće koriste za unos podataka s konzole i ispis na konzolu.

Učitavati podatke bajt po bajt nas neće previše usrećiti. U tom trenutku u pomoć dolazi klasa **BufferedReader**. Pojednostavljeno rečeno, klasa **BufferedReader** omogućava unošenje čitavih linija teksta s ekrana, što će nam uvalike olakšati unos podataka.

Konstruktor klase **BufferedReader** je oblika **BufferedReader (Reader r)**. Kao što znamo, naš **System.in** je tipa **InputStream**, a ne **Reader**, međutim i za to se je netko pobrinuo. **System.in** ćemo transformirati u **Reader** koristeći **InputStreamReader**.

Pojednostavljeno rečeno, želimo li kreirati objekt koji će nam omogućiti čitanje linija teksta sa standardnog ulaza, to ćemo napraviti na sljedeći način:

```
BufferedReader br = new BufferedReader (new InputStreamReader (System.in))
```

Izvršavanjem navedenog dijela koda bit će kreiran objekt *br* koji će biti znakovni stream, a omogućit će nam unos podataka sa standardnog ulaza.

Za unos jednog reda teksta sa standardnog ulaza koristit ćemo metodu **readLine ()** klase **BufferedReader**.

Ispis podataka na standardni izlaz znatno je jednostavniji. Ispis nam omogućavaju metode:

**public void print (String s)** – ispisuje string *s* na standardni izlaz i točku umetanja (kursor) ostavlja u istom redu, tako da sljedeći ispis započinje neposredno iza ispisanog stringa  
**public void println (String s)** – ispisuje string *s* na standardni izlaz i točku umetanja (kursor) premješta u novi red, tako da sljedeći ispis započinje u novom redu.

### Primjer 12 – 1:

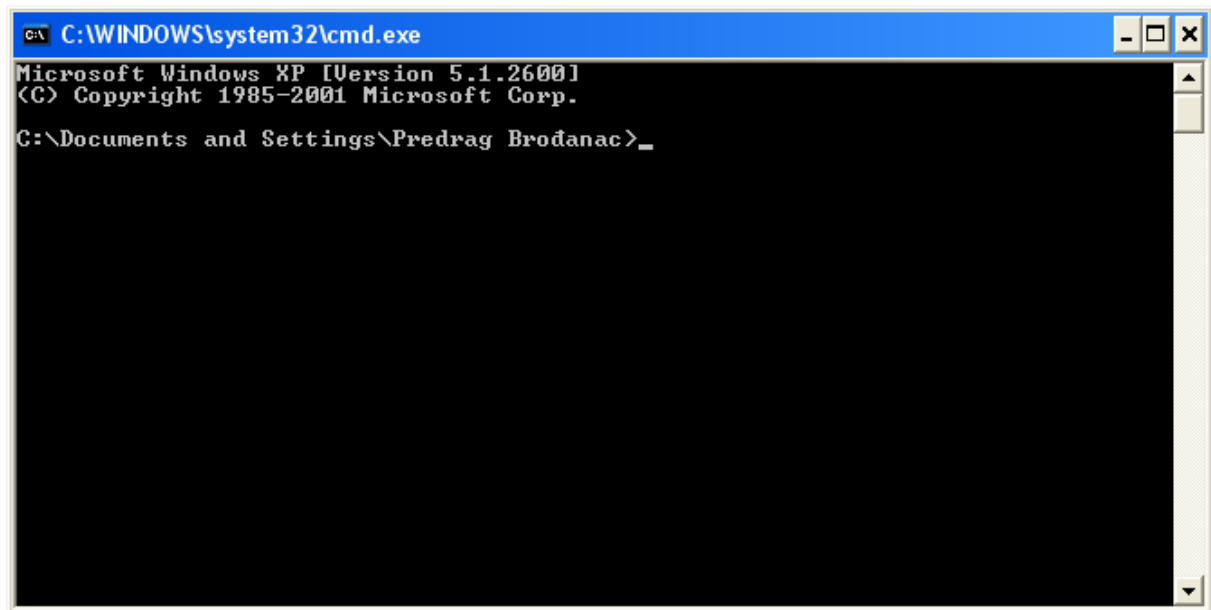
Napišimo konzolni program koji će unositi dva broja sa standardnog ulaza i ispisivati njihov zbroj na standardni izlaz.

Rješenje:

```
import java.io.*;
public class Zbroj
{
    public static void main (String[] s)
    {
        try
        {
            BufferedReader br = new BufferedReader (new
            InputStreamReader (System.in));
            String tmp;
            System.out.print ("Unesi prvi broj: ");
            tmp = br.readLine ();
            int a = Integer.parseInt (tmp);
            System.out.print ("Unesi drugi broj: ");
            tmp = br.readLine ();
            int b = Integer.parseInt (tmp);
            int c = a + b;
            System.out.println ("Zbroj brojeva " + a + " i " + b + "
            je " + c);
        }
        catch (Exception e)
        {
            System.out.println (e.getMessage ());
        }
    }
}
```

Dani program ćemo izvršiti na sljedeći način:

- pokrenut ćemo komandni prompt, najlakše ćemo to napraviti tako da kliknemo na gumb **Start**, te odaberemo **Run**. U okvir koji se pojavi upišemo *cmd* te kliknemo na gumb **OK**, otvorit će se komandni prompt, kao na slici 12 – 2.



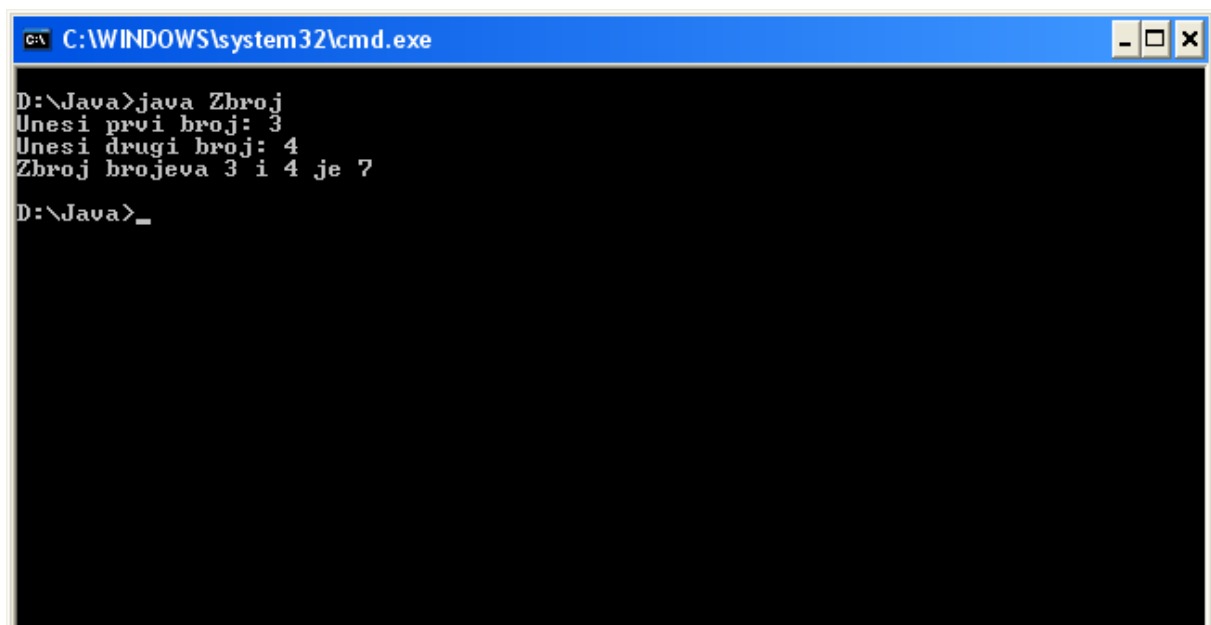
**Slika 12 – 2:** Komandni prompt

- premjestimo se na disk i u mapu gdje nam se nalazi klasa *Zbroj*, npr, ako se nalazi u *D:\Java* pisat ćemo:  
**>d:**  
**>cd Java**
- upisat ćemo:  
**>java Zbroj**

**Napomena:**

Ovdje je bitno voditi računa o velikim i malim slovima kod imena klase. Ako se npr. klasa zove *Zbroj*, a mi smo upisali npr. **java zbroj**, doći će do greške.

- unosit ćemo podatke kako se od nas traži



**Slika 12 – 3:** Izvršavanje programa **Zbroj** iz komandnog prompta

## Tekstualne datoteke

U nastavku ćemo naučiti kako rezultate obrade, odnosno podatke općenito spremiti u datoteke te ih naknadno pročitati i s njima raditi.

Za čitanje i pisanje podataka u datoteke postoji nekoliko klasa, mi ćemo koristiti sljedeće:

- **FileReader** – za učitavanje podataka iz datoteke
- **FileWriter** – za spremanje podataka u datoteku.

### Čitanje podataka iz datoteka

Kao što smo već rekli za čitanje podataka iz tekstualnih datoteka koristit ćemo klasu **FileReader**. Prilikom kreiranja instance klase **FileReader** kao parametar konstruktora možemo proslijediti puni naziv datoteke iz koje ćemo čitati podatke (uključujući i put do datoteke). U tom slučaju će kreiranje instance klase **FileReader** imati sljedeći oblik:

```
FileReader fin = new FileReader ("naziv_datoteke")
```

npr. ako je datoteka spremljena na disku *D* pod nazivom *podaci.txt*, tada će konstruktor imati oblik:

```
FileReader fin = new FileReader ("d:\\podaci.txt");
```

#### Napomena:

Primijetimo da smo kod naziva datoteke koristili `\\` a ne `\` koji inače koristimo za označavanje puta do neke mape. Razlog tome jest činjenica da se znak `\` koristi za označavanje posebnih znakova (*vidi poglavlje 2*).

Da bismo mogli čitati podatke iz neke datoteke ta datoteka uistinu mora postojati na definiranoj lokaciji. Ukoliko datoteka ne postoji pojaviti će se odgovarajuća greška. Kako bismo izbjegli pojavljivanje grešaka dio programa koji čita podatke iz datoteke uvijek ćemo staviti u **try/catch** blok.

**FileReader** je znakovni stream, te ćemo stoga za čitanje jednog znaka koristiti metodu **read ()**. Svaki puta kada se pozove, metoda **read ()** iz datoteke pročita sljedeći znak i vraća njegovu odgovarajuću cjelobrojnu vrijednost (kôd). Kôd znaka ćemo konvertirati u znak tako da napravimo konverziju cjelobrojnog tipa (`int`) u znakovni (`char`), odnosno da napravimo *cast* tipova.

#### Primjer 12 – 2:

Napišimo program koji će kreirati prozor s jednim okvirom za tekst. Učitavanjem programa se u okvir za tekst treba prepisati tekst iz tekstualne datoteke *tekst.txt*.

#### Rješenje:

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
public class Prijepis extends JFrame
{
    private final int sirina = 220;
```

```

private final int visina = 100;
private Container c;
private JTextField t;
public Prijepis()
{
    setTitle ("Prijepis");
    setSize (sirina, visina);
    setDefaultCloseOperation (EXIT_ON_CLOSE);

    c = getContentPane ();
    c.setLayout (null);
    String s = "";
    int i;
    try
    {
        FileReader fin = new FileReader ("d:\\tekst.txt");
        do
        {
            i = fin.read ();
            s = s + (char) i;
        }
        while (i != -1);
    }
    catch (Exception e)
    {
        s = ("Dokument ne postoji");
    }

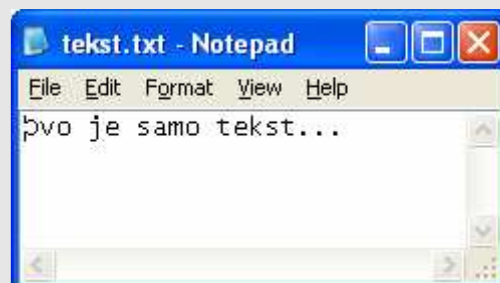
    t = new JTextField ();
    t.setSize (200, 25);
    t.setLocation (5, 20);
    t.setText (s);
    c.add (t);

    setVisible (true);
}

public static void main (String[] s)
{
    Prijepis p = new Prijepis ();
}
}

```

Ukoliko smo u tekstualni dokument *tekst.txt* upisali sljedeći sadržaj:



pokretanjem programa ćemo na ekranu dobiti sljedeće:



Čitanje teksta iz ulazne datoteke znak po znak nije nešto što će nas učiniti sretnim, puno više bi nas usrećila mogućnost čitanja riječi po riječi ili čitavog retka iz datoteke. U tu svrhu ćemo i ovdje koristiti klasu **BufferedReader** i njenu metodu **readLine ()**, koja vraća string koji sadrži učitani redak iz datoteke.

Slično kao kod standardnog unosa, konstruktoru klase **BufferedReader** proslijedit ćemo parametar koji je tipa **FileReader** a predstavlja dokument iz kojeg ćemo čitati podatke.

### Primjer 12 – 3:

Izmijenimo prethodni primjer (12 – 2) tako da pročita cijeli redak te pročitani string zapiše u odgovarajući okvir za tekst.

#### Rješenje:

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
public class Prijepis extends JFrame
{
    private final int sirina = 220;
    private final int visina = 100;
    private Container c;
    private JTextField t;
    public Prijepis()
    {
        setTitle ("Prijepis");
        setSize (sirina, visina);
        setDefaultCloseOperation (EXIT_ON_CLOSE);

        c = getContentPane ();
        c.setLayout (null);
        String s = "";
        try
        {
            FileReader fin = new FileReader ("d:\\tekst.txt");
            BufferedReader redak = new BufferedReader (fin);
            s = redak.readLine ();
        }
        catch (Exception e)
        {
            s = ("Dokument ne postoji");
        }

        t = new JTextField ();
        t.setSize (200, 25);
        t.setLocation (5, 20);
        t.setText (s);
        c.add (t);

        setVisible (true);
    }

    public static void main (String[] s)
    {
```



```

        Prijepis p = new Prijepis ();
    }
}

```

### Napomena:

Budući da objekt *fin*, koji je instanca klase **FileReader** koristimo samo kao parametar konstruktora klase **BufferedReader**, ovdje smo, kao što smo to radili kod standardnog unosa, mogli izostaviti njegovo kreiranje. Stoga smo dio programa:

```

FileReader fin = new FileReader ("d:\\tekst.txt");
BufferedReader redak = new BufferedReader (fin);

```

mogli zamijeniti sa:

```

BufferedReader redak = new BufferedReader (new FileReader ("d:\\tekst.txt"
));

```

## Spremanje podataka u datoteke

Za spremanje podataka u tekstualne dokumente koristit ćemo klasu **FileWriter**. Slično kao i kod učitavanja podataka i ovdje ćemo koristiti još jednu pomoćnu klasu **PrintWriter**. Za upis podataka u tekstualne datoteke, slično kao i za upis na standardni izlaz, koristit ćemo metode **print (String s)** i **println (String s)**.

Jedan od konstruktora klase **PrintWriter** ima parametar tipa **FileWriter**, koji predstavlja dokument u koji upisujemo podatke.

Za razliku od čitanja dokumenata gdje je neophodno morao postojati dokument iz kojeg čitamo podatke, prilikom pisanja podataka dokument ne mora nužno postojati i u tom će se slučaju kreirati automatski, tj. kreirat ćemo ga kreiranjem instance klase **FileWriter**.

Da bismo podatke koje smo upisali u dokument uistinu i spremili u taj dokument moramo pozvati metodu **close ()** nad objektom tipa **PrintWriter**.

### Primjer 12 – 4:

Izmijenimo prethodni primjer (12 – 3) tako da mu dodamo još jedan gumb *Spremi* klikom na koji će se tekst iz tekstualnog okvira spremiti u tekstualni dokument.

#### Rješenje:

```

import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.awt.event.*;
public class Prijepis extends JFrame implements ActionListener
{
    private final int sirina = 220;
    private final int visina = 150;
    private Container c;
    private JTextField t;
    private JButton b;
    public Prijepis()
    {
        setTitle ("Prijepis");
        setSize (sirina, visina);
        setDefaultCloseOperation (EXIT_ON_CLOSE);
    }
}

```

```
c = getContentPane ();
c.setLayout (null);
String s = "";
try
{
    FileReader fin = new FileReader ("d:\\tekst.txt");
    BufferedReader readak = new BufferedReader (fin);
    s = readak.readLine ();
}
catch (Exception e)
{
    s = ("Dokument ne postoji");
}

t = new JTextField ();
t.setSize (200, 25);
t.setLocation (5, 20);
t.setText (s);
c.add (t);

b = new JButton ();
b.setText ("Spremi");
b.setSize (200, 25);
b.setLocation (5, 50);
b.addActionListener (this);
c.add (b);

setVisible (true);
}

public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == b)
        spremiti ();
}

public void spremiti ()
{
    try
    {
        FileWriter fout= new FileWriter ("d:\\tekst.txt");
        PrintWriter pw = new PrintWriter (fout);
        pw.print (t.getText ());
        pw.close();
    }
    catch (Exception e)
    {
        t.setText ("Ne mogu spremiti dokument");
    }
}

public static void main (String[] s)
{
    Prijepis p = new Prijepis ();
}
}
```

**Napomena:**

Slično kao i kod upisivanja podataka u dokument, sljedeći dio koda:

```
FileWriter fout = new FileWriter ("d:\\tekst.txt");
```

```
PrintWriter pw = new PrintWriter (fout);
```

mogli smo zamijeniti sa:

```
PrintWriter pw = new PrintWriter (new FileWriter ("d:\\tekst.txt" ));
```

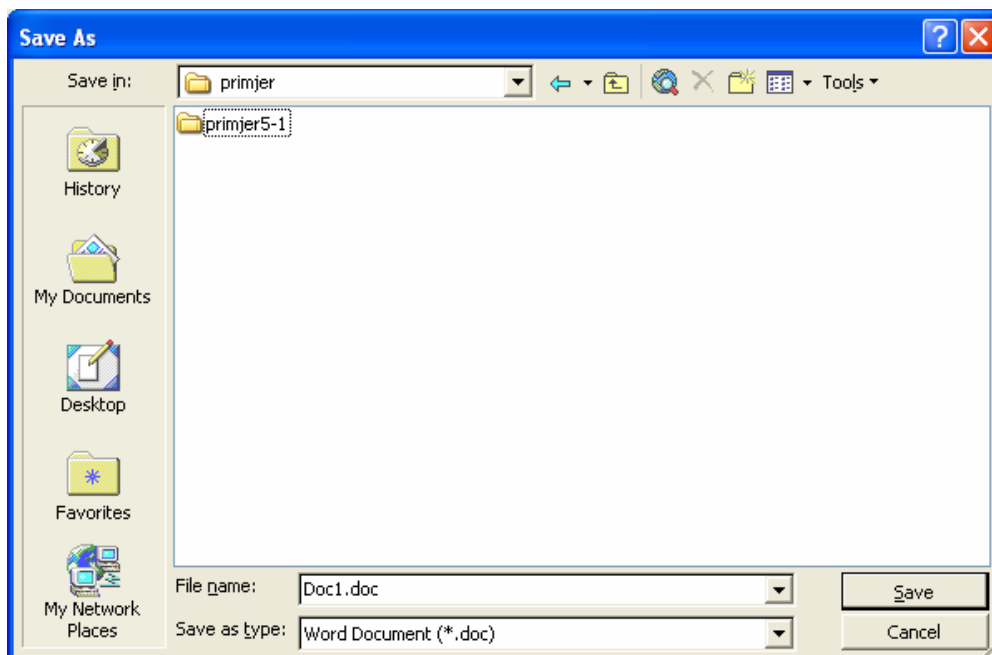


# 13

## Dijaloški prozori

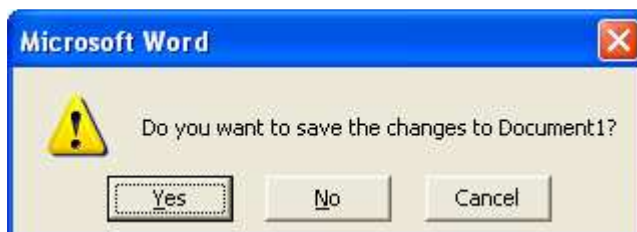
- Dijaloški prozori za otvaranje/spremanje dokumenata
- Poruke, potvrde i jednostavan unos
- Klasa `Color` i odabir boja
- Klasa `Font`

Dijaloške prozore svakodnevno susrećemo u radu s računalom. Npr. želimo li spremiti neki dokument u Wordu otvorit će nam se dijaloški prozor:



**Slika 13 – 1:** Prozor za spremanje dokumenata u OS Windows

Nadalje, želimo li npr. izaći iz Worda, a da prije toga nismo spremili napisani dokument otvorit će nam se dijaloški prozor:



**Slika 13 – 2:** Dijaloški prozor MS Worda

Dijaloški prozori su u stvari već gotovi prozori za neke često korištene radnje (spremanje dokumenata, otvaranje dokumenata, ispisivanje kratkih poruka,...).

## Dijaloški prozori za otvaranje/spremanje dokumenata

Dijaloške prozore za otvaranje i spremanje dokumenata kreirat ćemo posebnom klasom **JFileChooser**.

Metode klase **JFileChooser**, koje ćemo najviše koristiti su:

```
public int showOpenDialog (Component c) – otvara dijaloški prozor za
otvaranje dokumenta. Parametar c odnosi se na matičnu komponentu dijaloškog prozora i mi
ćemo tu uvijek pisati this. Vrijednost koju vraća ova metoda je neka od cjelobrojnih konstanti
klase JFileChooser:
- JFileChooser.CANCEL_OPTION – ako je pritisnut gumb Cancel
- JFileChooser.APPROVE_OPTION – ako je pritisnut gumb OK
- JFileChooser.ERROR_OPTION – ako je došlo do greške
public int showSaveDialog (Component c) – otvara dijaloški prozor za
spremanje dokumenta.
public File getSelectedFile () – vraća odabrani dokument.
```

**File** je posebna klasa u Javi. Dvije metode klase **File** koje ćemo najčešće koristiti su:

```
public String getName () – vraća naziv dokumenta
public String getPath () – vraća puno ime dokumenta uključujući i put do
dokumenta.
```

### Primjer 13 – 1:

Napišimo program koji će kreirati prozor s tri okvira za tekst i dva gumba. U okvire za tekst će se unositi ime, prezime te starost osobe, dok će na gumbima pisati *Spremi* i *Otvori*. Klikom na gumb *Spremi* treba se otvoriti dijaloški prozor u kojem će se odabrati ime dokumenta pod kojim će se spremiti podaci, a klikom na gumb *Save* dijaloškog prozora podaci će se spremiti u odabrani dokument (svaki podatak u svoj red). Klikom na gumb *Otvori* treba se otvoriti dijaloški prozor za odabir dokumenta, a odabirom dokumenta će se podaci iz prva tri njegova reda upisati u odgovarajuće okvire za tekst na prozoru.

#### Rješenje:

```
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.awt.event.*;
public class SOPodaci extends JFrame implements ActionListener
{
    private final int sirina = 280;
    private final int visina = 210;
    private Container c;
    private JTextField t1, t2, t3;
    private JLabel l1, l2, l3;
    private JButton b1, b2;

    public SOPodaci()
    {
        setTitle ("Podaci");
        setSize (sirina, visina);
        setDefaultCloseOperation (EXIT_ON_CLOSE);

        c = getContentPane ();
        c.setLayout (null);
```

```

        l1 = new JLabel ();
        l1.setText ("Ime");
        l1.setSize (100, 25);
        l1.setLocation (5, 5);
        c.add (l1);

        t1 = new JTextField ();
        t1.setSize (150, 25);
        t1.setLocation (110, 5);
        c.add (t1);

        l2 = new JLabel ();
        l2.setText ("Prezime");
        l2.setSize (100, 25);
        l2.setLocation (5, 40);
        c.add (l2);

        t2 = new JTextField ();
        t2.setSize (150, 25);
        t2.setLocation (110, 40);
        c.add (t2);

        l3 = new JLabel ();
        l3.setText ("Starost");
        l3.setSize (100, 25);
        l3.setLocation (5, 75);
        c.add (l3);

        t3 = new JTextField ();
        t3.setSize (150, 25);
        t3.setLocation (110, 75);
        c.add (t3);

        b1 = new JButton ();
        b1.setText ("Otvori");
        b1.setLocation (5, 120);
        b1.setSize (125, 25);
        b1.addActionListener (this);
        c.add (b1);

        b2 = new JButton ();
        b2.setText ("Spremi");
        b2.setLocation (135, 120);
        b2.setSize (125, 25);
        b2.addActionListener (this);
        c.add (b2);

        setVisible (true);
    }

    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == b1)
            otvori ();
        else
            spremi ();
    }

    public void spremi ()
    {

```

```
        try
        {
            JFileChooser fc = new JFileChooser ();
            if (fc.showSaveDialog (this) == JFileChooser.APPROVE_OPTION)
            {
                String fname =fc.getSelectedFile ().getPath();
                FileWriter fout = new FileWriter (fname);
                PrintWriter pw = new PrintWriter (fout);
                pw.println (t1.getText ());
                pw.println (t2.getText ());
                pw.println (t3.getText ());
                pw.close();
            }
        }
        catch (Exception e)
        {
        }
    }

    public void otvori ()
    {
        try
        {
            JFileChooser fc = new JFileChooser ();
            if (fc.showOpenDialog (this) == JFileChooser.APPROVE_OPTION)
            {
                String fname = fc.getSelectedFile().getPath();
                FileReader fin = new FileReader (fname);
                BufferedReader redak = new BufferedReader (fin);
                t1.setText (redak.readLine ());
                t2.setText (redak.readLine ());
                t3.setText (redak.readLine ());
            }
        }
        catch (Exception e)
        {
        }
    }

    public static void main (String[] s)
    {
        SOPodaci p = new SOPodaci ();
    }
}
```

---

## Poruke, potvrde i jednostavan unos

Kreiranje dijaloških prozora koji će korisniku omogućiti unos vrijednosti, ispisati poruke upozorenja,... omogućit će nam metode klase **JOptionPane**.





Klasa **JOptionPane** ima ukupno četiri metode za prikaz dijaloških prozora:

- **showMessageDialog**
- **showConfirmDialog**
- **showInputDialog**
- **showOptionDialog**

Sve navedene metode su static metode, što znači da ćemo ih pozivati na razini klase *JOptionPane*.



Navedene metode dolaze s nekoliko opcionalnih parametara:

- *roditeljska komponenta* – naziv prozora koji iz kojeg se otvara dijaloški prozor, mi ćemo najčešće kao ovaj parametar koristiti **this**;
- *poruka* – tekst poruke koja će biti ispisana na prozoru;
- *naslov* – tekst koji će se ispisati na naslovnoj traci prozora;
- *opcije* – jedna od sljedećih kombinacije gumba:
  - DEFAULT\_OPTION
  - YES\_NO\_OPTION
  - YES\_NO\_CANCEL\_OPTION
  - OK\_CANCEL\_OPTION
- *sličica poruke* – jedna od sljedećih sličica:
  -  - QUESTION\_MESSAGE
  -  - INFORMATION\_MESSAGE
  -  - WARNING\_MESSAGE
  -  - ERROR\_MESSAGE
- *niz objekata* – najčešće se radi o stringovima koji su u stvari tekstovi na gumbima kod nekih vrsta dijaloških prozora;
- *označeni objekt* – jedan od objekata, iz prijašnjih parametara, koji će biti inicijalno označen.

Nazivi parametara *opcija* i *sličica prozora* su u stvari cjelobrojne konstante klase *JOptionPane*. Kao i same metode, ove parametre ćemo pozivati direktno iz klase, dakle ispred imena parametra pisat ćemo *JOptionPane*.

## Metoda `showMessageDialog`

`ShowMessageDialog` metodu koristit ćemo kada za kreiranje najjednostavnijeg oblika dijaloškog prozora, u slučajevima kada korisniku želimo prikazati prozor s jednostavnom porukom (obavijest, upozorenje, poruka o grešci,...). Na prozoru se uvijek nalazi samo jedan gumb i to je gumb OK.


Metodu možemo pozvati na više načina, najčešće korišteni su:

```
void showMessageDialog (Component roditeljska_komponenta,  
Object poruka);
```

ili

```
void showMessageDialog (Component roditeljska_komponenta,  
Object poruka, String naslov, int opcije, int sličica_poruke);
```

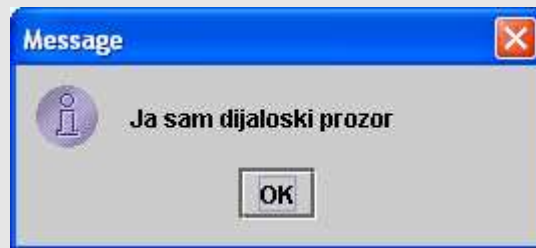
Kao što je i za pretpostaviti u prvom ćemo slučaju dobiti prozor kojem ćemo specificirati samo poruku, dok će sličica i naslov biti inicijalni. Naslov će u tom slučaju biti *Message*, dok će sličica

poruke biti .

Npr. izvršavanjem metode:

```
JOptionPane.showMessageDialog (this, "Ja sam dijaloski prozor");
```

na ekranu će biti prikazan sljedeći prozor:

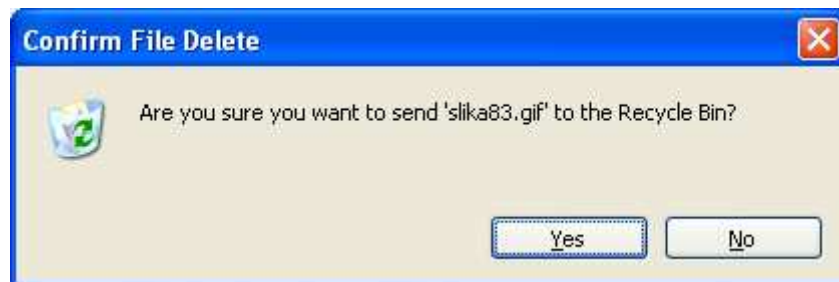


Primijetimo da smo isti prozor na ekranu mogli dobiti i na sljedeći način:

```
JOptionPane.showMessageDialog (this, "Ja sam dijaloski prozor", "Message",  
JOptionPane.INFORMATION_MESSAGE);
```

## Metoda showConfirmDialog

Metodu showConfirmDialog koristimo u slučajevima kada želimo da korisnik potvrdi ili poništi neku akciju. Npr. kod brisanja dokumenta u operativnom sustavu Windows pojavit će se sljedeći gumb:



Slika 13 – 3: Dijaloški prozor *Confirm*

Ovdje se radi o tzv. *Confirm* prozoru. Ukoliko korisnik klikne na gumb *Yes* dokument će biti izbrisan, inače, ukoliko klikne na gumb *No* dijaloški prozor će se zatvoriti a dokument neće biti izbrisan.

Kao i metoda showMessageDialog i showConfirmDialog može dolaziti s različitim brojem parametara, najčešće varijante poziva ove metode su:


```
int showConfirmDialog (Component roditeljska_komponenta,  
Object poruka);
```

ili

```
int showMessageDialog (Component roditeljska_komponenta,  
Object poruka, String naslov, int opcije);
```

ili

```
int showMessageDialog (Component roditeljska_komponenta,
Object poruka, String naslov, int opcije, int sličica_poruke);
```

Inicijalni naslov ovog dijaloškog prozora je: *Select an option*; inicijalna sličica poruke je: , dok će, ako drugačije nije određeno na prozoru biti tri gumba: *Yes*, *No* i *Cancel*.

Budući da ovaj dijaloški prozor nudi korisniku da odabere jednu od opcija, klikom na odgovarajući gumb, razumno je postaviti si pitanje: a kako ćemo znati na koji je gumb korisnik kliknuo?

U tu svrhu metoda `showMessageDialog` vraća cjelobrojnu vrijednost koja je u stvari oznaka gumba na koji je korisnik kliknuo. Već smo prije rekli da je svaka kombinacija gumba koja se može pojaviti na prozoru (`YES_NO_OPTION`,...) u stvari cjelobrojna konstanta. Isto tako postoje cjelobrojne konstante za gumbе:

gumb	konstanta	broj
OK	OK_OPTION	0
Yes	YES_OPTION	0
No	NO_OPTION	1
Cancel	CANCEL_OPTION	2

**Tablica 13 – 1:** Gumbi i odgovarajuće konstante

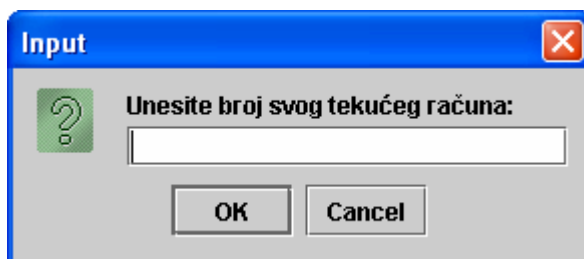
To znači da ćemo metodu `showConfirmDialog` u programima pozivati npr. na sljedeći način:

```
int n = JOptionPane.showConfirmDialog (this, "Jeste li sigurni?");

if (n == JOptionPane.YES_OPTION)
    //kod ako je pritisnut gumb OK
else if (n == JOptionPane.NO_OPTION)
    //kod ako je pritisnut gumb NO
else
    //kod ako je pritisnut gumb Cancel
```

## Metoda `showInputDialog`

Metodu `showInputDialog` koristit ćemo kada od korisnika tražimo unos neke jednostavne vrijednosti. Za razliku od svih dosadašnjih dijaloških prozora, ovaj prozor ima još i okvir za unos teksta.



**Slika 13 – 4:** Dijaloški prozor *input*

Kao i sve prijašnje, metoda `showInputDialog` je preopterećene (overloadana) te dolazi u više varijanti, najčešće su:

```
String showInputDialog (Component roditeljska_komponenta,
Object poruka);
String showInputDialog (Component roditeljska_komponenta,
Object poruka, String naslov, int sličica_poruke);
String showInputDialog (Component roditeljska_komponenta,
Object poruka, String tekst);
```

pri čemu je **tekst** inicijalna poruka koja će se pojaviti u okviru za tekst

Hm, a što je s tekstom koji smo unijeli u okvir za tekst? Imalo bi smisla nekako doći do njega. U tu svrhu metoda `showInputDialog` vraća string i taj string je u stvari tekst koji je unesen u okvir za tekst.

Stoga ćemo ovu metodu najčešće koristiti na sljedeći način:

```
String poruka = JOptionPane.showInputDialog (this, "Unesite broj svog
tekućeg računa:");
```

## Metoda `showOptionDialog`

Ova metoda programeru omogućava prilagođavanje izgleda prozora. Tako je moguće na gumbе dodati vlastiti tekst, izmijeniti sličicu prozora,...

Jedina varijanta poziva ove metode je:

```
int showOptionDialog (Component roditeljska_komponenta, Object
poruka, String naslov, int opcije, int sličica_poruke,
ImageIcon proizvoljna_sličica, Object[] proizvoljne_opcije,
Object inicijalna_opcija);
```

Prvih šest parametara su standardni parametri i susreli smo ih kod prethodnih vrsta dijaloških prozora, dok preostala 3 redom predstavljaju:

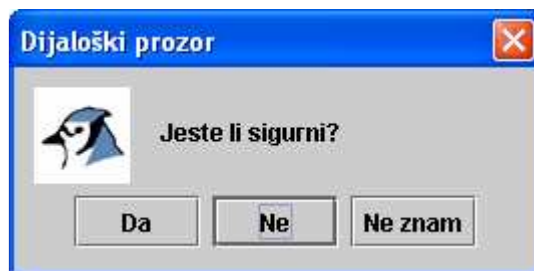
- **proizvoljna sličica** – ukoliko nam niti jedna od postojećih sličica poruke (`QUESTION_MESSAGE`, `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`) ne odgovara, možemo kao sličicu poruke postaviti neku svoju proizvoljnu sličicu. Sličica je u stvari objekt tipa **ImageIcon**. Jedan od konstruktora klase **ImageIcon** ima parametar ime slike, koju želimo dodijeliti objektu. Ime slike obuhvaća čitav put do slike, npr. *D:\Java\slika.jpg*;
- **proizvoljne opcije** – na ovaj način je moguće definirati vlastiti tekst na gumbima. Proizvoljne opcije su u stvari niz objekata, koji su u stvari stringovi koji predstavljaju tekstove koji će biti ispisani na gumbima i to onim redom kako su poslagnani u nizu. Gumbova će na dijaloškom prozoru biti onoliko koliko ima proizvoljnih opcija;
- **inicijalna opcija** – predstavlja proizvoljnu opciju koja će biti inicijalno označena.

Metoda `showOptionDialog` vraća redni broj gumba na kojeg je korisnik kliknuo.

Npr. izvršavanjem sljedećeg dijela koda:

```
String [] tmp = {"Da", "Ne", "Ne znam"};
ImageIcon ic = new ImageIcon ("bluej-icon.gif");
int n = JOptionPane.showOptionDialog (this, "Jeste li sigurni?", "Dijaloški
prozor", JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE, ic, tmp,
tmp[1]);
```

rezultat će sljedećim dijaloškim prozorom:



Slika 13 – 5: Prilagođeni dijaloški prozor

Kao što možemo primijetiti tekstovi na gumbima su definirani u nizu `tmp` koji ima tri elementa: stringove *Da*, *Ne* i *Ne znam*.

Sličicu poruke smo definirali u objektu `ic`, koji je instanca klase `ImageIcon`. Sličica se nalazi u direktoriju u kojem se nalazi i aplikacija, stoga pri njenom kreiranju nije potrebno navoditi put do nje.

Posljednji parametar metode `showOptionDialog` je `tmp[1]`, što znači da će inicijalno biti označen drugi gumb (gumb na kojem piše *Ne*).

Klikne li korisnik na gumb *Da* metoda će vratiti 0, klikom na *Ne* vratit će 1, dok će klikom na *Ne znam* metoda vratiti 2. Klikne li korisnik na gumb *Close* (u gornjem desnom vrhu prozora), metoda će vratiti -1.

---

## Klasa `Color` i odabir boja

Svaka boju na ekranu moguće je dobiti kombinacijom triju osnovnih boja: crvene, zelene i plave. Miješanjem određenih količina ovih triju boja možemo dobiti milijune različitih nijansi boja. Količina svake od ovih triju boja je cijeli broj između 0 i 255.

Za sada ćemo predstavljati boju kao trojku brojeva  $(r, g, b)$ , pri čemu je:  $r$  – količina crvene boje;  $g$  – količina zelene boje; dok je  $b$  – količina plave boje.

U tom slučaju će npr.  $(255, 0, 0)$  biti crvena boja (maksimalno crvene, ništa zelene, ništa plave),  $(255, 255, 0)$  – žuta boja (kombinacija crvene i zelene),  $(0, 0, 0)$  – crna boja,  $(255, 255, 255)$  – bijela boja,...

Za rad s bojama Java posjeduje klasu **Color**. Jedan od konstruktora klase **Color** je oblika:

```
public Color (int r, int g, int b) – kreira instancu boje s količinom crvene
boje r, zelene g i količinom plave boje b.
```

Primjerice, sljedećom naredbom:

```
Color col = new Color (0, 0, 255);
```

bit će kreiran objekt `col` koji će predstavljati plavu boju.

Na ovaj način kreiran objekt (instanca) boje može se dodati većini elemenata grafičkog korisničkog sučelja (gumbima, labelama, okviru,...) kao pozadinska boja ili kao boja teksta, u tu svrhu većina elemenata grafičkog korisničkog sučelja ima sljedeće dvije metode:

```
void setForeground (Color c) – definira boju teksta na elementu  
void setBackground (Color c) – definira boju pozadine elementa
```

Osim na ovaj način, neke je boje moguće definirati i implicitno, navođenjem imena boje. U stvari se radi o static konstantama definiranim u klasi **Color**. Definiranjem boje na ovaj način nije potrebno kreirati instancu klase **Color** već je boju moguće definirati s **Color.konstanta**, pri čemu je konstanta jedna od sljedećih konstanti:

□□□□□□□□	□□□□□□□□□□
<b>Color.black</b>	(0, 0, 0)
<b>Color.blue</b>	(0, 0, 255)
<b>Color.green</b>	(0, 255, 0)
<b>Color.cyan</b>	(0, 255, 255)
<b>Color.darkGray</b>	(64, 64, 64)
<b>Color.gray</b>	(128, 128, 128)
<b>Color.lightGray</b>	(192, 192, 192)
<b>Color.red</b>	(255, 0, 0)
<b>Color.magenta</b>	(255, 0, 255)
<b>Color.pink</b>	(255, 175, 175)
<b>Color.orange</b>	(255, 200, 0)
<b>Color.yellow</b>	(255, 255, 0)
<b>Color.white</b>	(255, 255, 255)

Tablica 13 – 2: Predefinirane konstante za neke boje

Želimo li npr. zadati da je boja teksta na gumbu crvena, to možemo napraviti na sljedeći način:

```
...  
private JButton b;  
...  
Color col = new Color (255, 0, 0);  
b.setForeground (col);  
...
```

ili

```
...  
private JButton b;  
...  
b.setForeground (Color.red);  
...
```

Kombinacijom količina crvene, zelene i plave boje, uistinu je moguće dobiti milijune boja, no na ovaj način dobiti baš onu boju koju želimo, nije niti malo jednostavno. U tu svrhu postoji dijaloški prozor za odabir boja. Klasa koja omogućava kreiranje ovakvog dijaloškog prozora je **JColorChooser**. U osnovi nam treba samo jedna metoda ove klase, metoda **showDialog**,

**showDialog** je static metoda klase `JColorChooser`, što znači da se poziva na razini klase. Metoda `showDialog` će kreirati dijaloški prozor za odabir boja i vratit će objekt tipa `Color`, koji predstavlja odabranu boju.

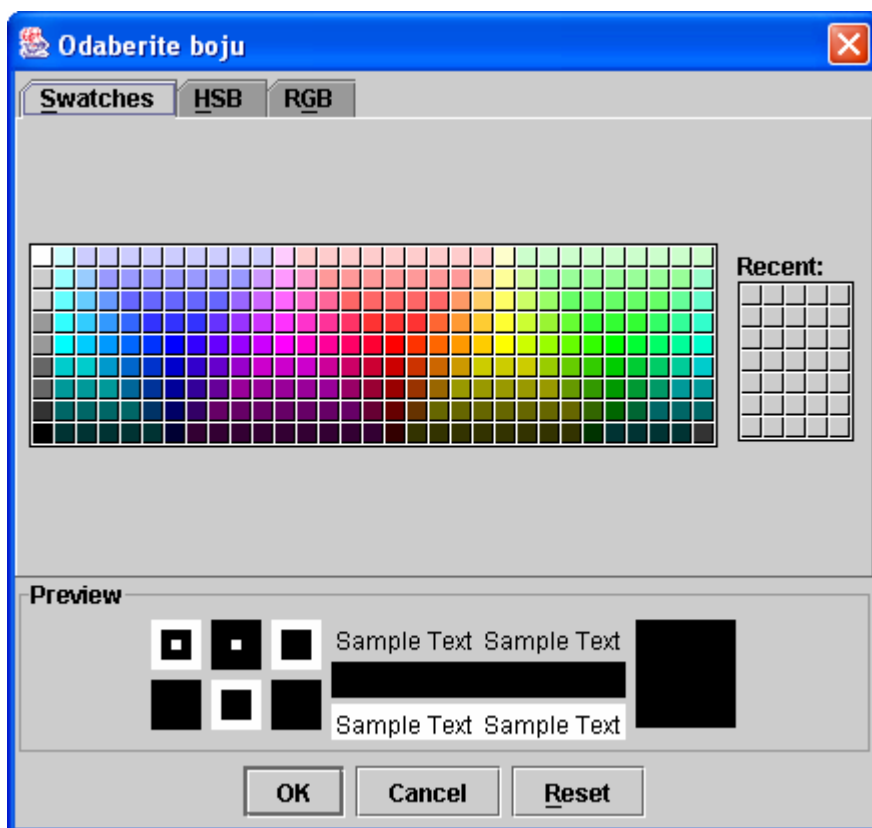
Opći oblik metode `showDialog` je sljedeći:

```
void showDialog (Component roditeljska_komponenta, Object  
naslov, Color inicijalna_boja);
```

Primjerice, izvršavanjem dijela programa:

```
Color c = JColorChooser.showDialog(this, "Odaberite boju", new Color (0, 0,  
0));
```

na ekranu će se pojaviti dijaloški prozor:



Slika 13 – 6: Dijaloški prozor za odabir boja

Kao što možemo primijetiti: u naslovnoj traci prozora je tekst "Odaberite boju", koji smo definirali kao drugi parametar metode `showDialog`, dok je inicijalno odabrana crna boja, što je definirano trećim parametrom (`new Color (0, 0, 0)`), koji u stvari predstavlja crnu boju.

---

## Klasa **Font**

Upravo smo naučili kako raditi s bojama, kako definirati boju teksta i pozadine na elementima GUI-a,... Razumno je zapitati se: kako definirati oblik ili veličinu znakova na nekom elementu? Naravno, programeri Java su se pobrinuli i za to. Postoji posebna klasa **Font** kojom je moguće definirati veličinu, stil i oblik znakova. Konstruktor klase **Font** je sljedećeg oblika:

```
public Font (String oblik, int stil, int veličina);
```

**oblik** je neki od naziva fontova: *Arial*, *Courier*, *Garamond*,...

**stil** znakova može biti **bold**, *italic*, plain ili kombinacija ***bold + italic***. Za stilove su definirane posebne cjelobrojne konstante u klasi **Font**:

- **BOLD**
- **ITALIC**
- **NORMAL**

Navedene konstante su static, što znači da ih možemo pozivati na razini klase. Želimo li da nam znakovi budu podebljani i nakošeni, pisat ćemo: **Font.BOLD + Font.ITALIC**.

**veličina** je prirodan broj koji predstavlja veličinu znakova u pixelima.

Npr. font čija će veličina biti 20, stil *bold* a oblik znakova *Arial* definirat ćemo na sljedeći način:

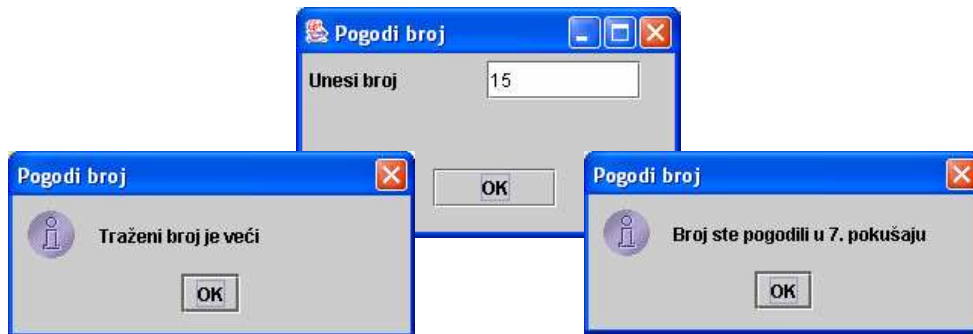
```
Font f = new Font ("Arial", Font.BOLD, 20);
```

Font nad nekim elementom GUI-a definirat ćemo metodom **setFont**.

```
public void setFont (Font f);
```

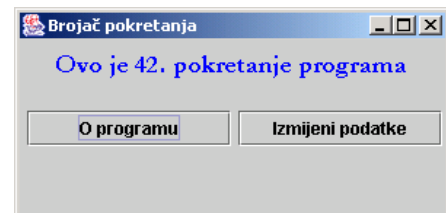


1. Napiši program koji će na ekranu crtati prozor kao na slici. Na početku program treba "zamisliti" slučajan prirodan broj do 100. Unosom broja u okvir za tekst i klikom na gumb OK treba se pojaviti dijaloški prozor s porukom da je traženi broj veći ili manji od unesenog, odnosno s porukom u kojem je pokušaju broj pogođen.

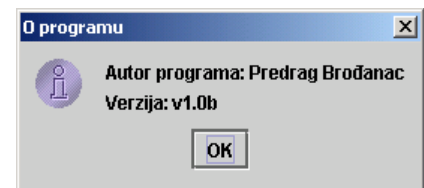


2. Napiši program koji će na ekranu iscrtavati prozor koji će se sastojati od dva gumba ("O programu" i "Izmijeni podatke") te jedne labela:

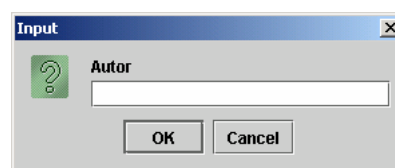
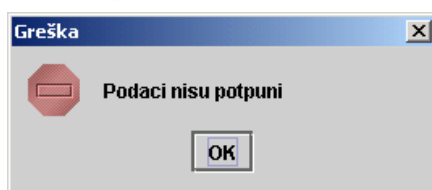
- a) Prilikom učitavanja programa se u labelu treba ispisati tekst: *Ovo je x. pokretanje programa*, pri čemu je **x** broj koji se učitava iz tekstualnog dokumenta (*brojac.txt*) i prilikom svakog pokretanja programa se povećava za 1.



- b) Klikom na gumb **O programu** u dijaloškom prozoru se trebaju ispisati podaci o autoru programa te verziji, a koji se nalaze u dokumentu *podaci.txt*. Ukoliko dokument ne postoji treba ispisati odgovarajuću grešku u dijaloškom prozoru.



- c) Klikom na gumb **Izmijeni podatke** trebaju se otvoriti dva dijaloška okvira za unos teksta (**Autor** i **Verzija**) te ukoliko su u dijaloške okvire uneseni podaci oni se trebaju spremiti u dokument *podaci.txt*. Ukoliko podaci nisu uneseni treba ispisati poruku o greški.





# 14

## Klasa Graphics

- Metode klase **Graphics**
- Crtanje grafova funkcija

Crtanje jednostavnih grafičkih elemenata, kao što su linije, elipse, pravokutnici,... omogućit će nam metode klase **Graph**. Klasa **Graph** sadržana je u paketu **java.awt.\***.

Sve što ćemo na prozoru crtati crtati ćemo u jednoj posebnoj metodi čije je zaglavlje oblika:

```
public void paint (Graphics g)
```

Kao što možemo primijetiti ova metoda ima kao parametar objekt **g** tipa **Graphics**, koji možemo zamisliti kao pozadinu na kojoj ćemo crtati. Nad objektom tipa **Graphics** ćemo pozivati odgovarajuće metode za crtanje te ćemo na taj način dobivati željene slike.

Metodu **paint ()** nije potrebno eksplicitno pozivati nigdje u programu. Program sam prilikom pokretanja traži postoji li metoda **paint ()**, te ukoliko postoji izvršava ju. Prostor za crtanje zauzima cijelu površinu ekrana. Prostor za crtanje sastoji se od mnoštva pixela (točkica). Svaki pixel ima svoje koordinate. Pixel koji se nalazi u gornjem lijevom vrhu prozora ima koordinate (0, 0), dok pixel u donjem desnom kutu ima koordinate (*širina\_prozora* - 1, *visina\_prozora* - 1). Na prozoru na kojem crtamo moguće je imati i sve ostale elemente grafičkog korisničkog sučelja. Jedini problem je što ne možemo odijeliti prozor na dio za elemente grafičkog sučelja i na dio za crtanje. Stoga nam se može dogoditi da npr. linije prelaze preko gumba,... Stoga je najbolje, ukoliko je izvedivo, na prozoru na kojem crtamo ne koristiti ostale elemente grafičkog korisničkog sučelja.

Svaki događaj na prozoru (resize, minimize,...) uzrokovat će ponovno izvršavanje metode **paint ()**. Isto tako je metodu **paint ()** moguće pozvati i u svakom drugom trenutku, npr. kao odgovor na neki događaj na nekom od elemenata grafičkog korisničkog sučelja, ako ih ipak imamo na prozoru. U tom slučaju ćemo metodu **paint ()** pozvati naredbom **repaint ()**.

#### Napomena:

**repaint ()** nikada nećemo pozivati u samoj metodi **paint ()**. To bi u stvari značilo da u metodi pozivamo samu metodu i tu bi ušli u beskonačno pozivanje.

## Metode klase Graphics

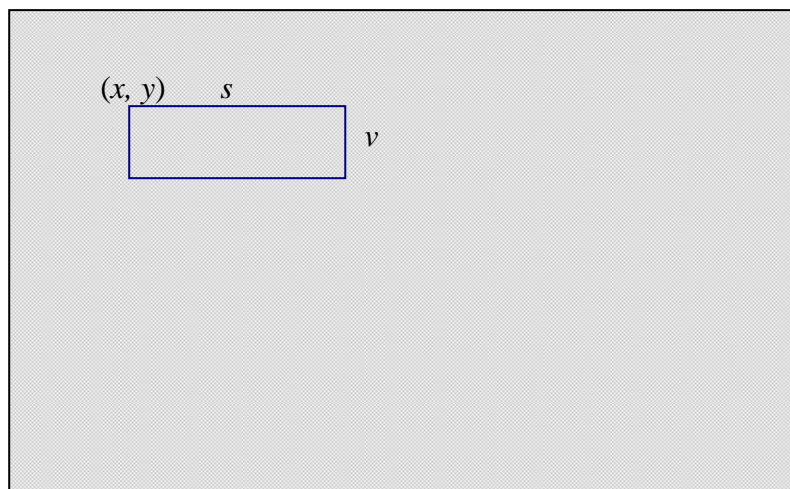
Najčešće metode za crtanje su:

```
public void drawLine (int x1, int y1, int x2, int y2)
```

Ova metoda crta liniju od točke  $(x1, y1)$  do  $(x2, y2)$  ekrana.

```
public void drawRect (int x, int y, int s, int v)
```

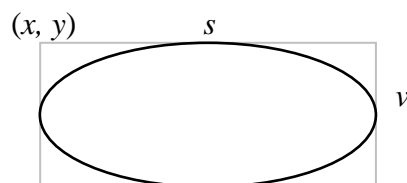
Crta pravokutnik čiji je gornji lijevi vrh u točki ekrana s koordinatama  $(x, y)$  dok mu je širina  $s$  a visina  $v$ .



Slika 14 – 1: Pravokutnik nacrtan metodom **drawLine** ( $x, y, s, v$ )

```
public void drawOval (int x, int y, int s, int v)
```

Crta elipsu unutar nevidljivog pravokutnika kojem je gornji lijevi vrh u točki s koordinatama  $(x, y)$ , dok mu je širina  $s$  a visina  $v$ .



Slika 14 – 2: Metoda **drawOval** ()

```
public void drawArc (int x, int y, int s, int v, int pk, int zk)
```

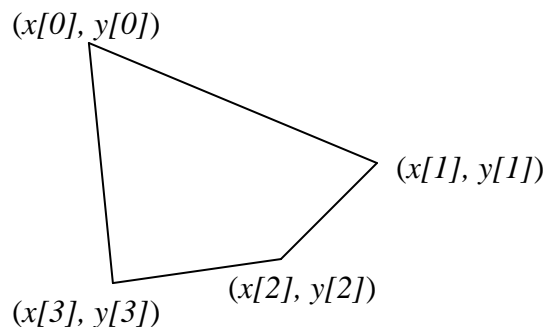
Crta dio luka elipse s početnim kutom  $pk$  do kuta  $zk$ .



Slika 14 – 3: Metoda **drawArc** ( )

```
public void drawPolygon (int[] x, int[] y, int n)
```

Crta poligon čiji su vrhovi točke s koordinatama  $(x[0], y[0])$ ,  $(x[1], y[1])$ , ...  $(x[n - 1], y[n - 1])$ , pri čemu je  $n$  broj vrhova poligona.



Slika 14 – 4: Metoda **drawPolygon** ( )

Navedene metode, osim **drawLine** ( ) crtaju samo rub odgovarajućeg lika. Analogno njima postoje metode koje crtaju odgovarajuće likove ispunjene bojom:

```
public void fillRect (int x, int y, int s, int v) – crta ispunjeni pravokutnik
public void fillOval (int x, int y, int s, int v) – crta ispunjenu elipsu
public void fillArc (int x, int y, int s, int v, int pk, int zk) – crta ispunjeni dio elipse
public void fillPolygon (int[] x, int[] y, int n) – crta ispunjeni poligon
```

Rekli smo kako crtati likove te kako crtati ispunjene likove, bilo bi zgodno znati kako definirati boju takvih likova. Boju bilo kojeg lika koji ćemo crtati definirat ćemo neposredno prije crtanja lika metodom:

```
public void setColor (Color b)
```

Postoji još jedna metoda klase **Graphics**, koju do sada nismo spomenuli, a koju ćemo često koristiti u svojim programima. Radi se o metodi:

```
public void drawString (String s, int x, int y)
```

Kao što je i za pretpostaviti, ova će metoda "crtati" string  $s$  na grafičkom ekranu. Možemo zamisliti da je string zapisan unutar nevidljivog pravokutnika i u tom slučaju gornji lijevi vrh tog pravokutnika ima koordinate  $(x, y)$ .

(x, y)

Ovo je neki tekst

Slika 14 – 5: Metoda `drawString ()`

Tekstu koji na ovaj način ispisujemo, slično kao i tekstu koji ispisujemo na ostalim elementima grafičkog korisničkog sučelja moguće je definirati oblik, veličinu,... metodom `setFont (Font f)`.

**Primjer 14 – 1:**

Napišimo program koji će na ekranu nacrtati olimpijske krugove.

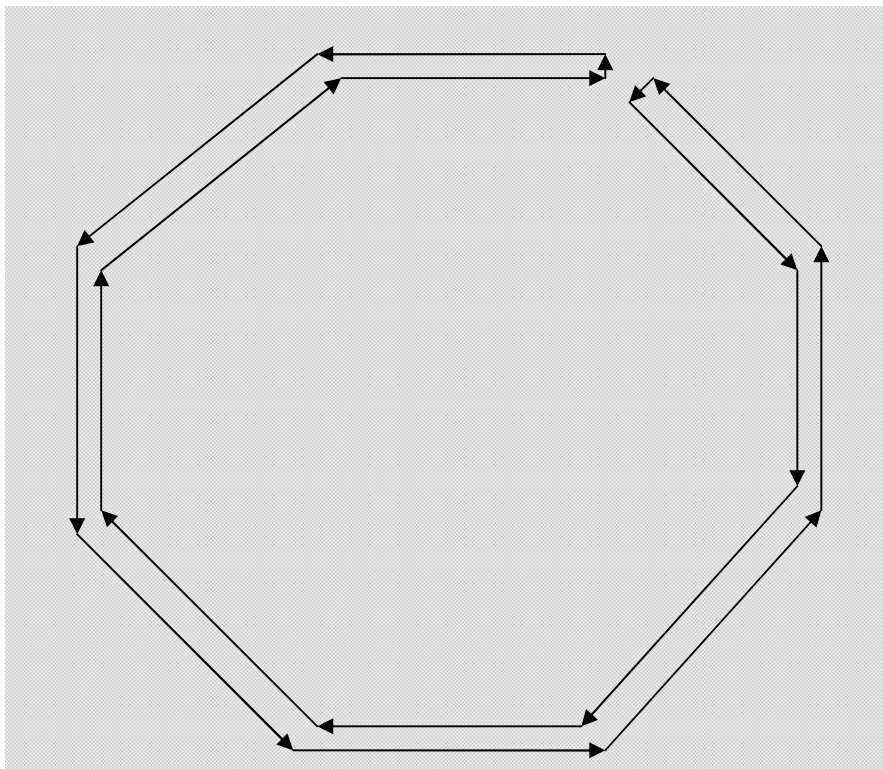
**Rješenje:**

Na prvi pogled bismo se mogli zapitati: Kakav je problem ovo nacrtati? Uistinu kada bismo htjeli napraviti 5 kružnica u odgovarajućim bojama, na odgovarajućim mjestima, uistinu ne bi bilo nikakvih problema. Međutim želimo li uistinu napraviti sliku što više nalik na gornju susrest ćemo se s nekoliko problema:

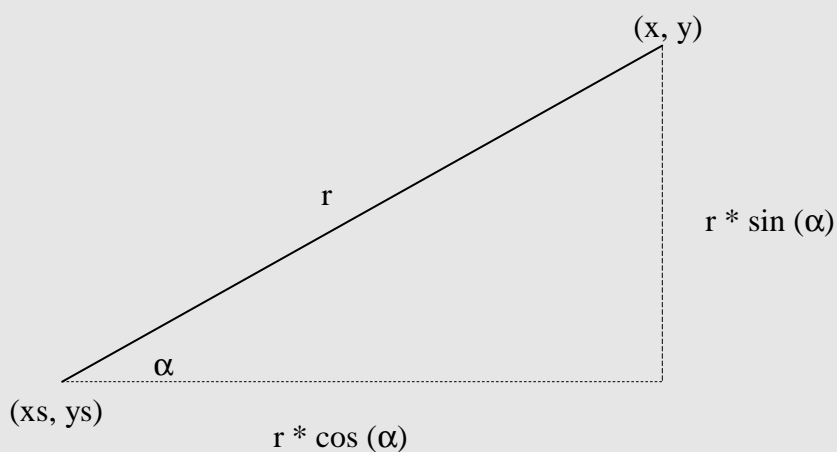
- kao što možemo primijetiti ne radi se o kružnicama već o tzv. kružnim vijencima (prstenima);
- kružni vijenci se međusobno isprepliću;
- ...

Kako napraviti kružni vijenac u nekoj boji? Znamo da klasa `Graphics` nema niti jednu metodu kojom bismo to eksplicitno napravili. Dakle trebamo smisliti neki drugi način. Pa vrlo jednostavno: želimo li npr. napraviti crveni kružni vijenac, jednostavno ćemo nacrtati crveni krug, a zatim ćemo napraviti njemu koncentričan bijeli krug nešto manjeg radijusa. To bi bilo sasvim uredu, međutim imajmo na umu da se naći krugovi isprepliću i da ćemo na taj način bijelim krugom obrisati i dio nekog drugog kruga. Dakle, ova ideja nam nije dobra.

Drugi, ispravniji način bio bi sljedeći: kružnicu možemo dobiti i kao pravilni  $n$ -terokut gdje je  $n$  dovoljno velik. Već ako stavimo da je  $n$  npr. 230, dobiveni 230-kut će biti potpuno nalik na kružnicu. Zašto  $n$ -terokut? Pa stoga što znamo da klasa `Graph` ima metodu `fillPoly ()` koja crta ispunjeni  $n$ -terokut. Mi bismo dakle htjeli napraviti  $n$ -terokut koji će u stvari izgledati kao kružni vijenac.



Da bismo nacrtali ovakav n-terokut, metodom **fillPoly ( )** trebaju nam koordinate vrhova n-terokuta, a za to će nam trebati malo matematike:



Dakle, ako je  $(x_s, y_s)$  "središte" n-terokuta, tada ćemo vrhove dobivati tako da mijenjamo veličinu kuta  $\alpha$ , te će točka  $(x, y)$  koja je vrh n-terokuta u tom slučaju imati koordinate:

$$x = x_s + r * \cos(\alpha)$$

$$y = y_s - r * \sin(\alpha)$$

Sada samo u stanju nacrtati n-terokut, međutim, zaboravili smo na još jedan problem, a to je činjenica da se kružni vijenci preklapaju. Upravo zbog toga nećemo moći sliku napraviti pomoću čitavih kružnih vijenaca. Stoga ćemo napisati metodu koja će pomoću n-terokuta crtati samo dio kružnog vijenca.



Metoda koja će crtati dio kružnog vijenca, sa središtem u točki  $(xs, ys)$ , radijusom veće kružnice  $r1$ , te radijusom manje kružnice  $r2$  od kuta  $pk$  do kuta  $zk$  je:

```
public void drawVijenac (Graphics g, int xs, int ys, int r1, int r2, int
pk, int zk)
{
    int[] x = new int [500];
    int[] y = new int [500];
    int i = 0, k = pk;
    //koordinata vrhova "većeg" n-terokuta
    while (k <= zk)
    {
        x [i] = (int) (xs + Math.cos (k / 180.0 * Math.PI) * r1);
        y [i] = (int) (ys - Math.sin (k / 180.0 * Math.PI) * r1);
        i++;
        k += 2;
    }
    //koordinata vrhova "manjeg" n-terokuta
    k-=2;
    while (k >= pk)
    {
        x [i] = (int) (xs + Math.cos (k / 180.0 * Math.PI) * r2);
        y [i] = (int) (ys - Math.sin (k / 180.0 * Math.PI) * r2);
        i++;
        k -= 2;
    }
    g.fillPolygon (x, y, i);
}
```

Nakon što smo napisali ovu metodu, samo crtanje tražene slike bit će jednostavno:

```
import java.awt.*;
import javax.swing.*;
public class Olympic extends JFrame
{
    public Olympic()
    {
        setTitle ("Olimpijski krugovi");
        setSize (550, 350);
        setDefaultCloseOperation (EXIT_ON_CLOSE);
        setBackground (Color.white);
        setVisible (true);
    }

    public void drawVijenac (Graphics g, int xs, int ys, int r1, int r2,
int pk, int zk)
    {
        int[] x = new int [500];
        int[] y = new int [500];
        int i = 0, k = pk;
        while (k <= zk)
```



```

        {
            x [i] = (int) (xs + Math.cos (k / 180.0 * Math.PI) * r1);
            y [i] = (int) (ys - Math.sin (k / 180.0 * Math.PI) * r1);
            i++;
            k += 2;
        }
        k-=2;
        while (k >= pk)
        {
            x [i] = (int) (xs + Math.cos (k / 180.0 * Math.PI) * r2);
            y [i] = (int) (ys - Math.sin (k / 180.0 * Math.PI) * r2);
            i++;
            k -= 2;
        }

        g.fillPolygon (x, y, i);
    }

    public void paint (Graphics g)
    {
        g.setColor (Color.blue);
        drawVijenac (g, 150, 150, 52, 44, 0, 360);
        g.setColor (Color.yellow);
        drawVijenac (g, 205, 200, 52, 44, 0, 360);
        g.setColor (Color.blue);
        drawVijenac (g, 150, 150, 52, 44, -20, 20);
        g.setColor (Color.black);
        drawVijenac (g, 260, 150, 52, 44, 0, 360);
        g.setColor (Color.yellow);
        drawVijenac (g, 205, 200, 52, 44, 70, 90);
        g.setColor (Color.green);
        drawVijenac (g, 315, 200, 52, 44, 0, 360);
        g.setColor (Color.black);
        drawVijenac (g, 260, 150, 52, 44, -20, 20);
        g.setColor (Color.red);
        drawVijenac (g, 370, 150, 52, 44, 0, 360);
        g.setColor (Color.green);
        drawVijenac (g, 315, 200, 52, 44, 70, 90);
    }

    public static void main (String[] s)
    {
        Olympic ol = new Olympic ();
    }
}

```

### **Napomena:**

Primijetimo da je metoda `drawVijenac ()` kao parametar između ostaloga imala i objekt tipa `Graphics`. Naime, radi se o činjenici objekt `g`, instanca klase `Graphics` postoji samo u metodi `paint ()`. Da bismo mu mogli pristupiti iz neke druge metode, moramo ga toj metodi proslijediti kao parametar.

### **Jeste li znali:**

Olimpijski krugove kao simbol olimpijskih igara kreirao je Baron Pierre de Coubertin 1913 godine. Pet krugova simboliziraju pet kontinenata na modernim olimpijskim igrama: Azija, Europa, Oceanija, Sjeverna Amerika i Južna Amerika. Zastava svake zemlje, s nekog od kontinenata, ima najmanje jednu od boja olimpijskih krugova. Boje krugova idući od lijeva na desno su: plava, žuta, crna, zelena i crvena.

## Crtanje grafova funkcija

Sa grafom funkcije po prvi se puta susrećemo već u matematici viših razreda osnovne škole. U nastavku ćemo naučiti kako crtati grafove funkcija u Javi. Kao što znamo, graf funkcije  $f$  u pravokutnom koordinatnom sustavu je skup točaka  $(x, f(x))$ , pri čemu je  $x$  bilo koja točka domene, dok je  $f(x)$  vrijednost funkcije  $f$  u točki  $x$ .

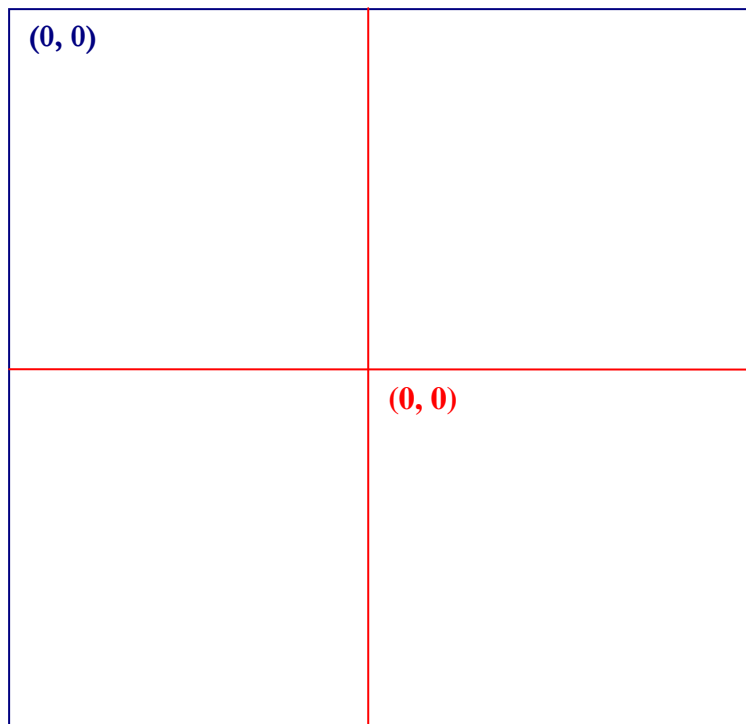
Promotrimo to sa aspekta programiranja. Mi bismo mogli graf funkcije prikazivati u jednom prozoru. Središte toga prozora bilo bi ishodište koordinatnog sustava. Nadalje nam trebaju koordinatne osi, što mogu biti dvije dužine paralelne s rubovima prozora te nam treba jedinična duljina, za koju možemo uzeti da je npr. 20 pixela.

Kao što smo rekli graf funkcije  $f$  je skup točaka  $(x, f(x))$  pravokutnog koordinatnog sustava. Dakle, da bismo u našem koordinatnom sustavu na prozoru, nacrtali graf funkcije, uzimat ćemo neke točke iz domene funkcije (točke s  $x$  osi) i za odgovarajuću točku domene računat ćemo vrijednost funkcije te ćemo na prozoru nacrtati točku.

U rečenom treba uočiti nekoliko bitnih činjenica:

- nema smisla uzimati bilo koje točke domene (točke s  $x$  osi), treba uzimati samo one koje su vidljive na ekranu;
- što uzmemo više točaka domene (s vidljivog dijela ekrana) to će graf funkcije biti precizniji;
- rekli smo da je ishodište koordinatnog sustava, točka s koordinatama  $(0, 0)$  središte ekrana, a s druge strane znamo da se točka prozora s koordinatama  $(0, 0)$  nalazi u gornjem lijevom kutu. Kako ćemo mi za crtanje koristiti metode klase `Graphics`, koje raspoznaju samo koordinate ekrana, a ne nikakvog našeg nacrtanog koordinatnog sustava, morat ćemo preračunavati koordinate. Isto tako bitno je voditi računa da je jedinična duljina na prozoru 1 pixel, a mi smo za jediničnu duljinu uzeli 20 pixela.

Razmotrimo malo posljednji problem te nađimo način na koji ćemo pretvarati koordinate našeg nacrtanog koordinatnog sustava u koordinatni sustav prozora. Radi ilustracije, zamislimo da prozor ima širinu i visinu 100 pixela. Želimo li koordinatni sustav na sredini ekrana, to znači da će ishodište tog sustava u stvari biti točka prozora s koordinatama  $(50, 50)$ .



Uzmimo npr. točku  $(1, 0)$  u nacrtanom koordinatnom sustavu, zanimaju nas koordinate te točke u koordinatnom sustavu prozora. Znamo da koordinate ishodišta koordinatnog sustava na prozoru imaju koordinate  $(50, 50)$ , znači pomaknemo li se za jednu jediničnu dužinu, nacrtanog koordinatnog sustava, u stvari se pomičemo za 20 pixela. Dakle točka  $(1, 0)$  nacrtanog koordinatnog sustava će u koordinatnom sustavu prozora imati koordinate  $(70, 0)$ . Lako se da zaključiti da će općenito točka s koordinatama  $(x, 0)$ , našeg zamišljenog koordinatnog sustava u koordinatnom sustavu prozora imati koordinate  $(50 + x * 20)$ .

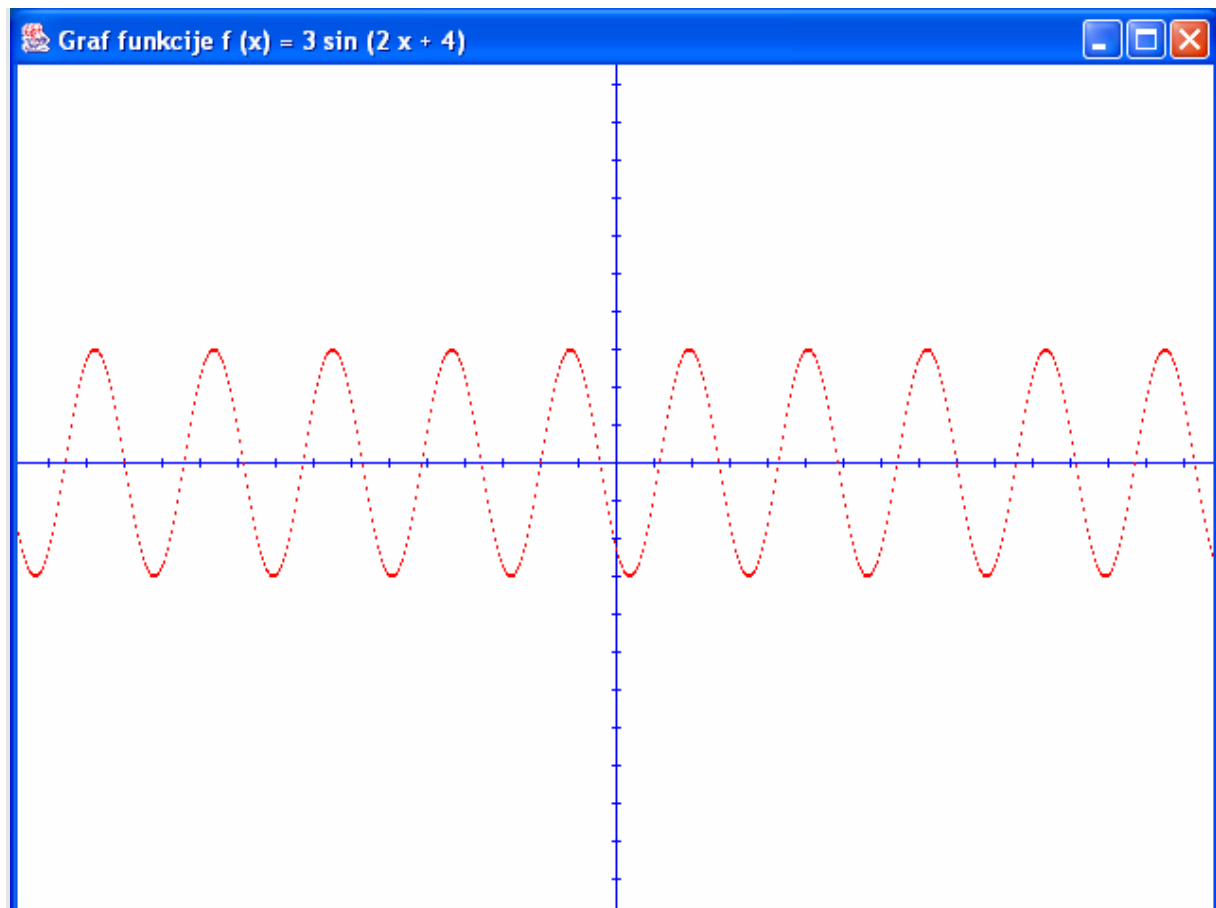
Nadalje, primijetimo da će npr.  $y$  koordinata točke  $(0, 1)$  nacrtanog koordinatnog sustava u koordinatnom sustavu prozora biti  $50 - 20$  pixela što iznosi 30. Dakle točka  $(0, 1)$  zamišljenog koordinatnog sustava će u koordinatnom sustavu prozora imati koordinate  $(0, 30)$ , odnosno općenito će točka s koordinatama  $(0, y)$  u nacrtanom koordinatnom sustavu, imati koordinate  $(0, 50 - y * 20)$  u koordinatnom sustavu prozora.

Općenito, neka je središte nacrtanog koordinatnog sustava točka s koordinatama  $(xs, ys)$  u koordinatnom sustavu prozora. Neka je jedinična dužina nacrtanog koordinatnog sustava 20 pixela, te neka je u njemu zadana točka s koordinatama  $(x, y)$ . Točka  $(x, y)$  će u koordinatnom sustavu prozora u tom slučaju imati koordinate  $(xp, yp)$ , pri čemu je:

$$\begin{aligned} xp &= xs + 20 * x \\ yp &= ys - 20 * y \end{aligned}$$

#### Primjer 14 – 2:

Napišimo program koji će unositi parametre  $a$ ,  $b$  i  $c$  funkcije  $f(x) = a \sin(bx + c)$ , te će crtati graf funkcije u koordinatnom sustavu čije će središte biti oko sredine ekrana veličine 640x480 pixela.



Rješenje:

```
import javax.swing.*;
import java.awt.*;
public class GrafF extends JFrame
{
    private final int SIRINA = 640, VISINA = 480;
    private int a, b, c;
    public GrafF()
    {
        try
        {
            a = Integer.parseInt(JOptionPane.showInputDialog (this,
                "Parametar a"));
        }
        catch (Exception e)
        {
            a = 1;
        }
        try
        {
            b = Integer.parseInt(JOptionPane.showInputDialog (this,
                "Parametar b"));
        }
        catch (Exception e)
        {
            b = 1;
        }
        try
        {
            c = Integer.parseInt(JOptionPane.showInputDialog (this,
```

```

        "Parametar c"));
    }
    catch (Exception e)
    {
        c = 0;
    }
    setTitle ("Graf funkcije f (x) = " + a + " sin (" + b + " x + "
+ c + ")");
    setSize (SIRINA, VISINA);
    setDefaultCloseOperation (EXIT_ON_CLOSE);
    setBackground (Color.white);
    setVisible (true);
}

private void xos (Graphics g)
{
    g.setColor (Color.blue);
    g.drawLine (0, VISINA / 2, SIRINA, VISINA / 2);
    for (int i = 1; i < SIRINA / 20; i++)
        g.drawLine (i * 20, VISINA / 2 - 2, i * 20, VISINA / 2 +
2);
}

private void yos (Graphics g)
{
    g.setColor (Color.blue);
    g.drawLine (SIRINA / 2, 0, SIRINA / 2, VISINA);
    for (int i = 1; i < VISINA / 20; i++)
        g.drawLine (SIRINA / 2 - 2, i * 20, SIRINA / 2 + 2, i *
20);
}

public void crtaj (Graphics g)
{
    double x = - SIRINA / 40.0, xs = SIRINA / 2.0, ys = VISINA /
2.0, y;
    int xp, yp;
    g.setColor (Color.red);
    while (x < SIRINA / 40)
    {
        y = a * Math.sin (b * x + c);
        xp = (int)(xs + 20 * x);
        yp = (int)(ys - 20 * y);
        g.drawOval (xp, yp, 1, 1);
        x += 0.05;
    }
}

public void paint (Graphics g)
{
    xos (g);
    yos (g);
    crtaj (g);
}

public static void main (String[] s)
{
    GrafF gF = new GrafF ();
}
}

```

Primijetimo da nam se graf funkcije sastoji os gustih točkica. Želimo li da nam graf bude gladak, da nema razmaka između točkica, točkice možemo spojiti linijama, i tom će slučaju metoda `crtaj ( )` imati sljedeći oblik:

```
public void crtaj (Graphics g)
{
    double x = - SIRINA / 40.0, xs = SIRINA / 2.0, ys = VISINA / 2.0, y;
    int xp, yp, xt, yt;
    y = a * Math.sin (b * x + c);
    xt = (int)(xs + 20 * x);
    yt = (int)(ys - 20 * y);
    g.setColor (Color.red);
    while (x < SIRINA / 40)
    {
        y = a * Math.sin (b * x + c);
        xp = (int)(xs + 20 * x);
        yp = (int)(ys - 20 * y);
        g.drawLine (xt, yt, xp, yp);
        xt = xp;
        yt = yp;
        x += 0.05;
    }
}
```



# 15

## Java appleti

- Primjer appleta
- Integriranje appleta u web stranicu
- Prosljeđivanje parametara appletu iz web browsera

Do sada smo se upoznali s dva tipa programa u Javi:

- konzolni programi, koje smo pokretali iz komandnog prompta, a vrijednosti smo im prosljeđivali preko standardnog ulaza ili iz datoteke
- programi s korisničkim sučeljem, kod kojih smo vrijednosti unosili preko elemenata grafičkog korisničkog sučelja ili iz fileova.

Na samo početku smo rekli da osim programa u Javi možemo raditi i applete. Radi se o posebnoj vrsti programa koja je integrirana u web stranicu i izvršava se unutar web browsera.

---

## Primjer appleta

Za razliku od programa s grafičkim korisničkim sučeljem, koje smo kreirali nasljeđujući klasu `JFrame`, kod appleta ćemo nasljeđivati klasu `JApplet`. Obje klase nalaze se u istom paketu `javax.swing`.

Prisjetimo se, kod programa smo uvijek imali konstruktor i metodu `main ()`. Kod appleta je stvar nešto drugačija, nema konstruktora i nema metode `main ()`. Ali zato će uvijek biti jedna od metoda:

**`public void init ()`** – poziva se iz web browsera ili applet viewera svaki puta kada se applet učita ili ponovno učita. Ova metoda na neki način zamjenjuje konstruktor, koji smo imali kod programa. Unutar nje je poželjno inicijalizirati svojstva appleta,...

**`public void paint (Graphics g)`** – za crtanje na appletu (analogno metodi `paint ()` kod programa)

osim navedenih metoda unutar appleta postoji i još niz drugih metoda, najčešće korištene su:

**`public void start ()`** – poziva se iz web browsera ili applet viewera, neposredno nakon pozivanja metode `init ()`, kada se kod appleta počinje izvršavati, te kada se stranica s appletom ponovo posjeti (bez da se gasi browser)

**`public void stop ()`** – poziva se iz browsera ili applet viewera svaki puta kada se prekida izvršavanje appleta (zatvaranje web stranice na kojoj se applet nalazi).

**`public void destroy ()`** – poziva se iz browsera ili applet viewera neposredno nakon metode `stop ()`.

**`public void showStatus ()`** – prikazuje poruku na statusnoj traci.

### Primjer 15 – 1:

Napišimo jednostavan applet koji će ispisivati poruku "Ja sam Java applet, a tko si ti?".

Rješenje:

```
import javax.swing.*;
import java.awt.*;
public class Poruka extends JApplet
{
    public void paint (Graphics g)
    {
        setBackground (Color.yellow);
        g.setFont (new Font ("Garamond", Font.BOLD, 24));
        g.setColor (Color.blue);
        g.drawString ("Ja sam Java applet, a tko si ti?", 20, 50);
    }
}
```



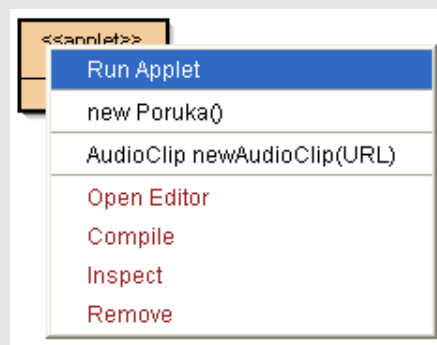
Ovako kreirani applet možemo pokrenuti na nekoliko načina. BlueJ razvojno okruženje nam omogućava izvršavanje appleta na dva načina:

- pokretanje u appletvieweru
- pokretanje u web browseru

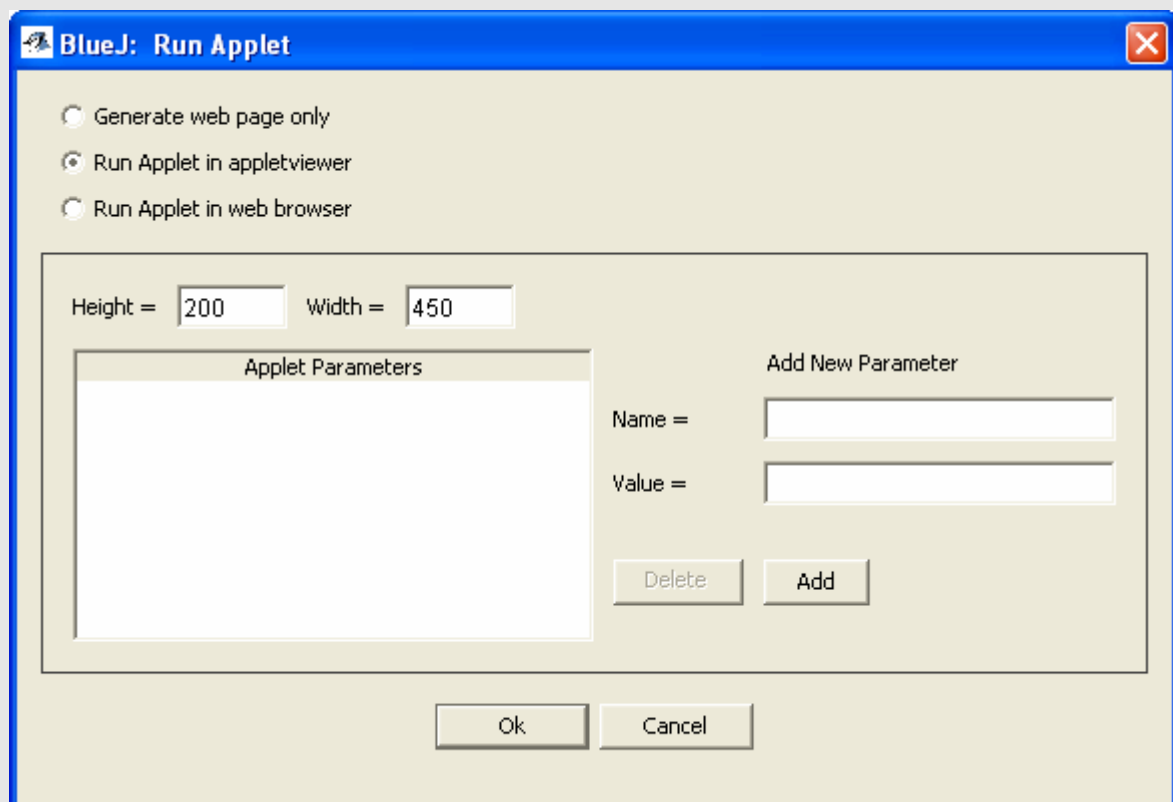
Na samom početku ćemo primijetiti da je već sličica appleta drugačija nego sličica programa.



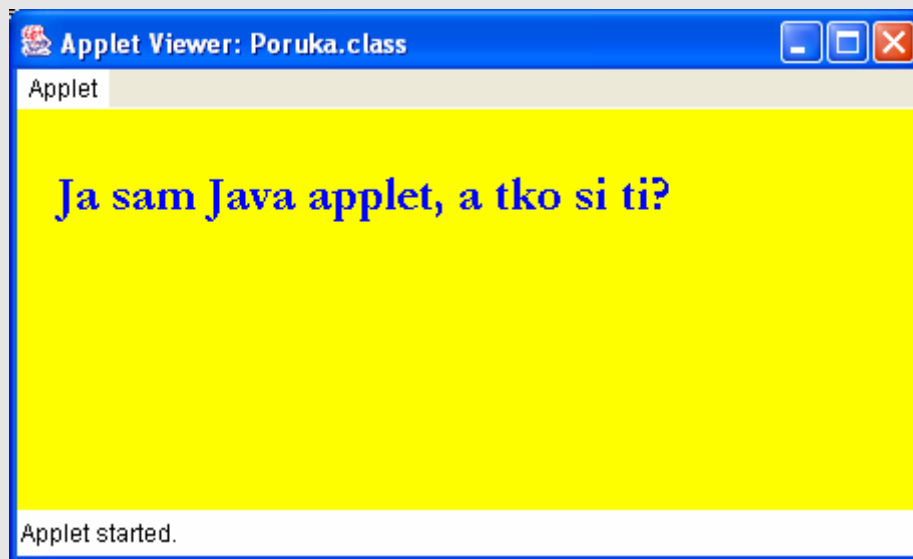
Klikom desnog gumba miša na sličicu appleta otvorit će nam se izbornik, kod kojeg ćemo odabrati **Run applet**:



Pokretanje appleta otvorit će nam se novi prozor u kojem ćemo moći definirati veličinu appleta, način na koji želimo pokrenuti applet, te dodati još neke parametre appletu, o kojima ćemo govoriti nešto kasnije.



Ovaj puta ćemo applet pokrenuti iz appletviewera, posebnog programa za pregledavanje appleta, definirat ćemo mu veličinu, te kliknuti na gumb OK i u posebnom prozoru će nam se otvoriti applet:



---

## Integriranje appleta u web stranicu

Kao što znamo, web stranica je multimedijalni dokument koji se nalazi negdje na Internetu. Između ostaloga web stranice mogu sadržavati i applete. U pozadini svake web stranice nalazi se HTML (*Hypertext Markup Language*). HTML je u stvari jezik za pisanje web stranica, sastoji se od tzv. tagova odnosno opisnika, pomoću kojih dizajniramo izgled stranice te stavljamo elemente na stranicu.

Svaka stranica ima sljedeći oblik:

```
<html>
<head>
...
</head>
<body>
...
</body>
</html>
```

Kao što možemo primijetiti, web stranica se sastoji od zaglavlja (head) i tijela (body) stranice. Unutar zaglavlja stranice nalazi se najčešće naslov stranice (onaj koji se vidi u naslovnoj traci web browsera), neke skripte,... dok se unutar tijela stranice nalazi sve ono što vidimo u browseru, pa ćemo onda unutar tijela staviti sam applet. Applet koji stavljamo na neku web stranicu mora biti kompajliran te moramo imati dokument s nastavkom *.class*. Tag za postavljanje appleta na stranicu je **<applet>** i ima sljedeću sintaksu:

```
<applet code = "ImeKlase.class" width = "sirina" height = "visina">
</applet>
```

Pri čemu je *ImeKlase.class* naziv klase koja generira applet, dok su *sirina* i *visina* vrijednosti za širinu i visinu appleta na web stranici.

Bitno je napomenuti da se sama klasa koja generira applet i web stranica trebaju nalaziti u istoj mapi, inače kod code treba navesti čitav put do filea u kojem se nalazi klasa.

### Primjer 15 – 2:

Kreirajmo web stranicu u koju ćemo uključiti applet napravljen u prethodnom primjeru.

Rješenje:

```
<html>
<head>
  <title>Stranica s Java appletom</title>
</head>
<body>
  Ispod ovog teksta nalazi se Java applet<br>
  <applet code = "Poruka.class" height = "200" width = "450">
  </applet>
</body>
</html>
```

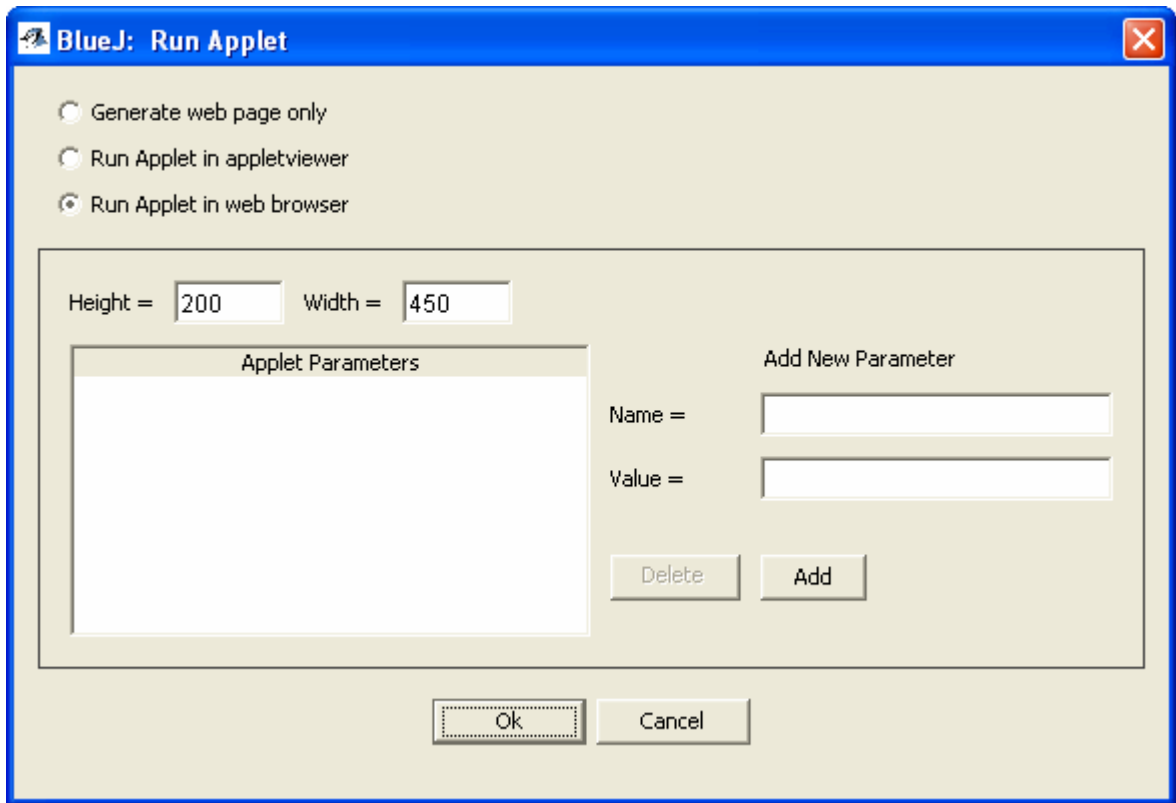
Ovaj kod možemo napisati u bilo kojem text procesoru (Notepad,...) spremimo ga npr. pod imenom *stranica.html* u mapu u kojoj se nalazi file s klasom *Poruka.class*. Tako kreiranu stranicu otvorimo s nekim web browserom i ukoliko je sve u redu u njemu ćemo dobiti sadržaj kao na sljedećoj slici:



Ovdje ima nekoliko HTML tagova koje nismo objasnili. Ukratko tagom <title> unutar zaglavlja definiramo naslov stranice (tekst koji se nalazi u naslovnoj traci preglednika). Tagom <br> naglašavamo da na tom mjestu želimo preći u novi red na stranici. Tekst neposredno prije taga <br> je običan tekst koji se ispisuje na stranici.

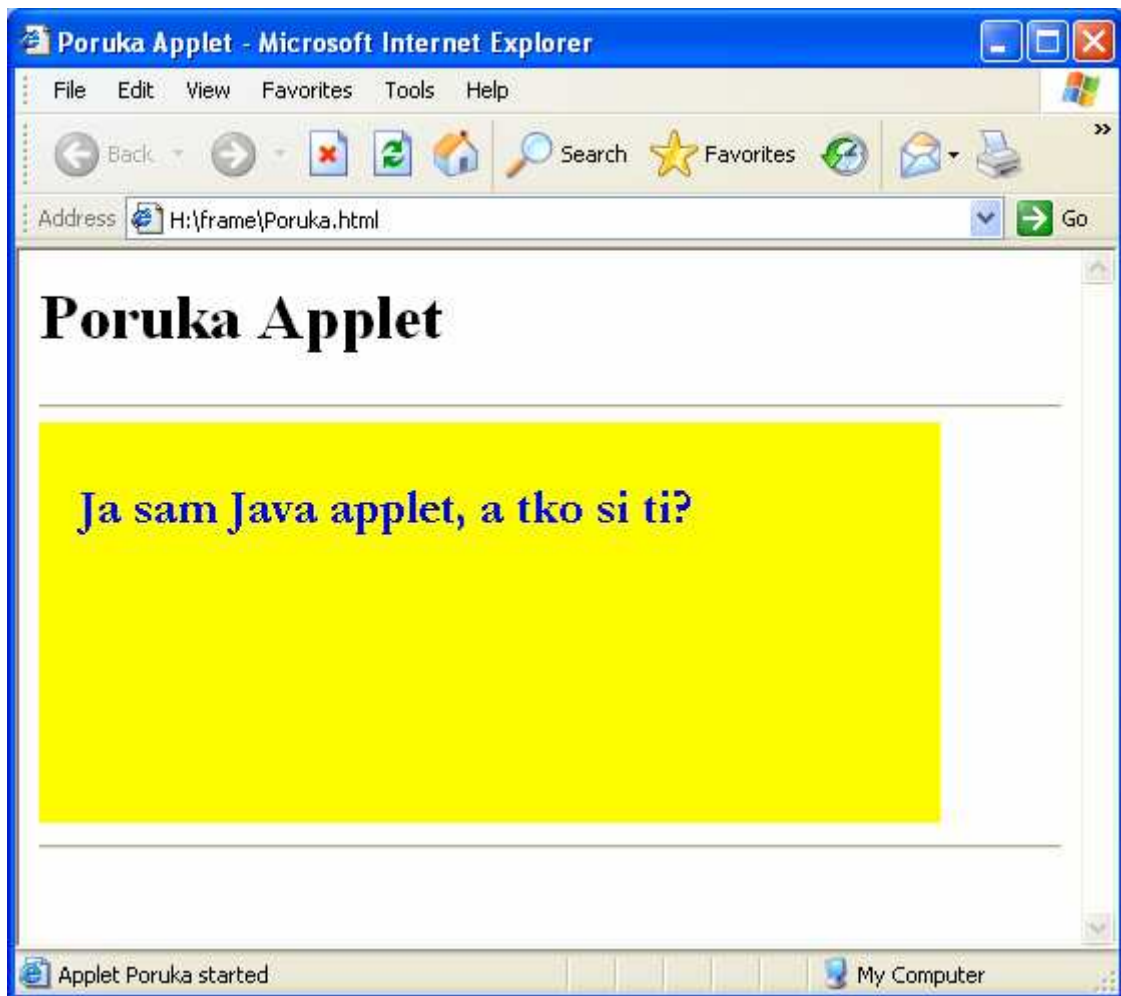
BlueJ razvojno okruženje nam nudi mogućnost automatskog kreiranja web stranice. Pomoću BlueJa ćemo stranicu kreirati na sljedeći način:

1. Kliknemo desnim gumbom miša na sličicu appleta za koji želimo da se nalazi na Web stranici.
2. Odaberemo ***Run Applet***.



3. U prozoru koji se pojavi odaberemo opciju ***Run Applet in web browser***, te upišemo širinu i visinu appleta unutar web stranice.
4. Kliknemo na gumb OK.

Otvorit će nam se web browser u kojem će se otvoriti stranica s appletom koji smo postavili unutar nje.



Stranica će biti kreirana u mapi u kojoj se nalazi i klasa koju smo pokretali kroz web browser. Kod ovako generirane stranice možemo pogledati tako da u izborniku **View** browsera odaberemo **Source**, i on u našem slučaju izgleda ovako:

```
<html>
<!-- This file automatically generated by BlueJ Java Development -->
<!-- Environment. It is regenerated automatically each time the -->
<!-- applet is run. Any manual changes made to file will be lost -->
<!-- when the applet is next run inside BlueJ. Save into a -->
<!-- directory outside of the package directory if you want to -->
<!-- preserve this file. -->
<head>
  <title>Poruka Applet</title>
</head>
<body>
  <h1>Poruka Applet</h1>
  <hr>
  <applet code="Poruka.class"
    width=450
    height=200
    codebase="."
    archive=""
    alt="Your browser understands the <APPLET> tag but isn't
    running the applet, for some reason."
  >
```

```

        Your browser is ignoring the <APPLET> tag!
    </applet>
    <hr>
</body>
</html>

```

**Napomena:**

Primijetimo da smo kod programa s grafičkim korisničkim sučeljem uvijek koristili metode: `setTitle()`, `setSize()`, `setDefaultCloseOperation()`, `setVisible()`,... kod appleta takvih metoda nema. Naime, nema smisla appletu dodavati naslov jer se nalazi unutar Web stranice kojoj se definira naslov. Veličina koju applet zauzima unutar prozora web preglednika, također je definirana na razini HTML-a, unutar taga `<applet>`. Applet pokrećemo neposredno s pokretanjem stranice, pa stoga nije potrebna metoda `main()`, koja se u stvari pokreće kada želimo pokrenuti neki Java program. Isto tako zatvaranjem stranice na kojoj se applet nalazi automatski se prekida i izvršavanje appleta.

## Prosljeđivanje parametara appletu iz web browsera

Kada smo govorili o pokretanju appleta iz BlueJ sučelja rekli smo da je appletima preko parametara moguće proslijediti neke vrijednosti. U nastavku ćemo govoriti nešto više o tome. Moglo bi se npr. dogoditi da je applet iz našeg primjera postane jako poželjan i svi ga poželes imati na svojim web stranicama. Međutim, ne sviđa se svima npr. tekst koji applet ispisuje ili možda ne vole svi žutu boju. Kako ne bismo morali svaki puta iznova raditi novi applet s drugim tekstom ili drugom bojom,... bilo bi zgodno kada bismo npr. boju ili tekst mogli definirati negdje izvan appleta. To uistinu i možemo napraviti i to preko parametara u HTML-u. Vrijednosti parametara ćemo u klasi koja generira applet pokupiti metodom:

```
String getParemater (String ime)
```

Unutar HTML-a ćemo parametre definirati tagom:

```
<param name = "ime" value = "vrijednost">
```

koji se nalazi unutar taga `<applet>`.

Prije nego što počnemo rješavati sljedeći primjer, upoznat ćemo se s još jednom klasom koju ćemo u njemu koristiti. Radi se o klasi **Date** za rad s datumima i vremenom.

Klasa **Date** ima nekoliko konstruktora, od kojih se najčešće koriste sljedeći:

**public Date ()** – kreira objekt tipa **Date**, pri čemu će datum i vrijeme biti inicijalizirani na trenutni datum i vrijeme  
**public Date (int g, int m, int d)** – kreira objekt tipa **Date**, pri čemu će vrijednost godine biti postavljena na **g**, mjeseca na **m** a dana na **d**. Godine se počinju računati od 1900, pa ako npr. želimo godinu 2005, vrijednost parametra **g** ćemo postaviti na 105. Mjeseci su definirani od 0-11. Pa ako želimo mjesec na npr. lipanj, vrijednost parametra **d** treba biti 5.  
**public Date (int g, int m, int d, int h, int min, int sec)** – kreira objekt tipa **Date** pri čemu će vrijednosti svojstava biti postavljene na prosljeđene vrijednosti, a godina i datum se definiraju kao kod prošlog konstruktora.

**public Date (long ms)** – kreira objekt tipa **Date**, pri čemu će datum i vrijeme biti postavljeni tako da predstavljaju datum koji dolazi **ms** sekundi nakon *1. siječnja 1900 00:00:00*.

Osim navedenih konstruktora često ćemo koristiti i neke od sljedećih metoda:

```
public int getYear () – vraća redni broj godine kao redni broj nakon 1900
public int getMonth () – vraća redni broj mjeseca, počevši od 0
public int getDate () – vraća dan u mjesecu
public int getDay () – vraća redni broj dana u tjednu
public int getHours () – vraća sate
public int getMinutes () – vraća minute
public int getSeconds () – vraća sekunde
public long getTime () – vraća vrijeme u milisekundama od 1. siječnja 1900 00:00:00
public void setYear () – postavlja godinu
public void setMonth () – postavlja mjesec
public void setDate () – postavlja dan u mjesecu
public void setDay () – postavlja redni broj dana u tjednu
public void setHours () – postavlja sate
public void setMinutes () – postavlja minute
public void setSeconds () – postavlja sekunde
public boolean equals (Date d1) – vraća true ako su datumi jednaki
public boolean after (Date d1) – vraća true ako je datum nakon specificiranog datuma d1
public boolean before (Date d1) – vraća true ako je datum prije specificiranog datuma d1
```

Klasa **Date** nalazi se u posebnom paketu **java.util**, koji ćemo morati uključiti u svoj program svaki puta kada ćemo raditi s datumima.

### Primjer 15 – 3:

Kreirajmo Java applet koji će unositi dan, mjesec i godinu rođenja neke osobe, te će na osnovu tih podataka računati bioritam osobe za razdoblje od broja dana koji će se zadati kao parametar kroz web stranicu. Bioritam se sastoji od 3 krivulje. Sve tri krivulje su periodičke, u trenutku rođenja su sve tri na 0. Krivulje su redom:

Fizički aspekt – period je 23 dana

Emotivni aspekt – period je 28 dana

Intelektualni aspekt – period je 33 dana

### Rješenje:

Applet:

```
import java.awt.*;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;
public class Bioritam extends JApplet implements ActionListener
{
    private Date dr, d = new Date ();
    private Container c;
    private JButton b;
    private JComboBox c1, c2, c3;
    private boolean initialized = false;
    private int broj_dana, k;
    public void init()
```

```

{
    //uzimanje vrijednosti parametra s web stranice
    broj_dana = Integer.parseInt (getParameter ("broj_dana"));
    k = 600 / broj_dana;
    c = getContentPane ();
    c.setLayout (null);

    c1 = new JComboBox ();
    c1.setSize (50, 25);
    c1.setLocation (10, 10);
    for (int i = 1; i <= 31; i++)
        c1.addItem (" " + i);
    c.add (c1);

    String[] mj = {"Siječanj", "Veljača", "Ožujak", "Travanj",
        "Svibanj", "Lipanj", "Srpanj", "Kolovoz", "Rujan", "Listopad",
        "Studenj", "Prosinac"};
    c2 = new JComboBox (mj);
    c2.setSize (150, 25);
    c2.setLocation (70, 10);
    c.add (c2);

    c3 = new JComboBox ();
    c3.setSize (70, 25);
    c3.setLocation (230, 10);
    for (int i = 1900; i <= 2005; i++)
        c3.addItem (" " + i);
    c.add (c3);

    b = new JButton ("OK");
    b.setSize (80, 25);
    b.setLocation (320, 10);
    b.addActionListener (this);
    c.add (b);
}

public void paint(Graphics g)
{
    super.paint (g);
    g.setColor(Color.white);
    g.fillRect(0, 100, 600, 520);
    if (initialized)
    {
        //starost osobe u danima kao razlika između dva datuma u
        //milisekundama, pa poslije pretvorena u dane
        long start = (long)((d.getTime () - dr.getTime ()) / (24 * 60 *
        60 * 1000));
        //crtanje krivulja
        crtaj (g, 28, Color.red, start);
        crtaj (g, 33, Color.blue, start);
        crtaj (g, 23, Color.green, start);
        koordinatni (g);
    }
}

//metoda crta graf za određeni aspekt na osnovu perioda ciklusa te
//starosti osobe u danima.
//Budući da se radi o periodičkom grafu, crtat ćemo ga kao sinusoidu
//s periodom b i to na intervalu od (start - broj_dana / 2)
//do (start + broj_dana / 2), pri čemu je start starost osobe u

```



```

//danim, na današnji datum, dok je broj_dana vrijednost parametra,
//proslijeđenog s web stranice a odnosi se na broj dana za koji
//želimo raditi bioritam.
public void crtaj (Graphics g, int b, Color c, long start)
{
    double x = start - broj_dana / 2, y;
    int xt = (int)(k * (x - start) + 300), yt = (int)(300 - 20 * 10 *
        Math.sin (2.0/b * Math.PI * x)), xx, yy;
    g.setColor (c);
    while (x < start + broj_dana / 2)
    {
        y = Math.sin (2.0/b * Math.PI * x);
        xx = (int)(k * (x - start) + 300);
        yy = (int)(300 - 20 * 10 * y);
        g.drawLine (xt, yt, xx, yy);
        xt = xx;
        yt = yy;
        x += 0.01;
    }
    if (b == 23)
        g.drawString ("Fizički ciklus", 10, 515);
    else if (b == 28)
        g.drawString ("Emotivni ciklus", 210, 515);
    else
        g.drawString ("Intelektualni ciklus", 410, 515);
}

//metoda crta koordinatni sustav ovisno o vremenskom intervalu za
//koji se bioritam radi.
public void koordinatni (Graphics g)
{
    g.setColor (Color.blue);
    g.drawLine (0, 300, 600, 300);
    g.drawLine (300, 100, 300, 500);
    Date tmp;
    int dan, danas = d.getDate ();
    String s;
    for (int i = -broj_dana / 2; i <= broj_dana / 2; i++)
    {
        g.drawLine (300 + i * k, 298, 300 + i * k, 302);
        tmp = new Date (d.getTime () + i * 24 * 60 * 60 * 1000);
        if (tmp.getDate () < 10)
            s = "0" + tmp.getDate ();
        else
            s = "" + tmp.getDate ();
        if (tmp.getMonth () + 1 < 10)
            s = s + ".0" + (tmp.getMonth () + 1);
        else
            s = s + "." + (tmp.getMonth () + 1);
        if (i % 2 == 0)
            g.drawString (s, 285 + i * k, 315);
    }
}

public void actionPerformed (ActionEvent e)
{
    initialized = true;
    dr = new Date (c3.getSelectedIndex (), c2.getSelectedIndex (),
        c1.getSelectedIndex () + 1);
    repaint ();
}

```

```

    }
}

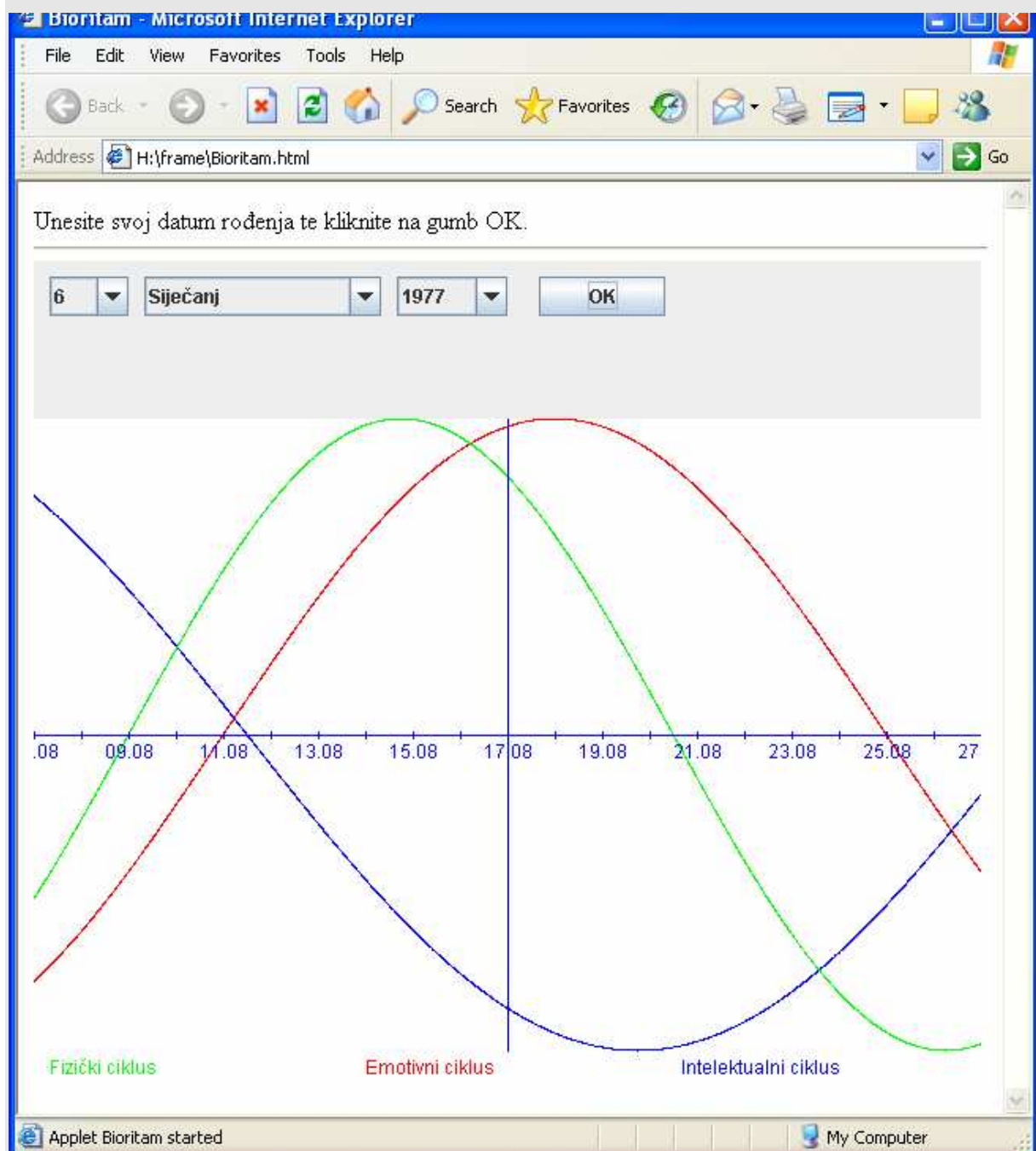
```

Web stranica:

```

<html>
<head>
    <title>Bioritam</title>
</head>
<body>
    Unesite svoj datum rođenja te kliknite na gumb OK.
    <hr>
    <applet code="Bioritam.class" width="600" height="520">
        <param name = "broj_dana" value = "20">
    </applet>
</body>
</html>

```





# 16

## Osnove mrežne komunikacije u Javi

- O računalnim mrežama
- Mrežna komunikacija u Javi
- Komunikacija putem socketa
- Višenitnost

Jedno od velikih područja upotrebe Jave je izrada mrežnih aplikacija, raznih servera,... U ovom ćemo se poglavlju, na stvarnom primjeru upoznati s osnovnim pojmovima vezanim uz mrežnu komunikaciju. Cilj nam je napraviti program koji će omogućiti komunikaciju korisnika na udaljenim računalima, tzv. chat.

## O računalnim mrežama

Mrežu čine dva ili više uređaja koja su međusobno spojena i mogu razmjenjivati podatke. Najčešći načini spajanja danas su kablovi odnosno wireless. Cilj mreže je prijenos podataka između uređaja unutar mreže tzv. čvorova. Čvorovi mogu biti računala, neki drugi tipovi hardwarea npr. printer ili neki mrežni uređaji zaduženi za pojedine segmente unutar komunikacije. Kako bi podaci stizali na ispravno odredište treba postojati pravilan način označavanja čvorova komunikacije. Svaki čvor unutar mreže ima svoju adresu. Baš isto kao što ako na razglednicu napišemo ispravnu adresu primatelja, bez obzira iz kojeg dijela svijeta ju slali ona će doći na odredište. Slična je situacija i u svijetu računala. Postoje stroga pravila komuniciranja tzv. **protokoli**. Svaki uređaj spojen na računalnu mrežu ima svoju jedinstvenu adresu. Ta adresa je fiksna i u principu ju nije moguće mijenjati.

Sam Internet, globalna računalna mreža, funkcionira na sličnom principu. Komunikacija putem Interneta temelji se na IP adresama. IP adresa se sastoji od 32 bita (4 puta po 8 bitova) i oblika je:

**bbb.bbb.bbb.bbb**

pri čemu je **bbb** neki nenegativan cijeli broj između 0 i 255. Primjer IP adrese je: 195.29.220.4. Spajanjem na Internet svako računalo dobije svoju jedinstvenu IP adresu i na taj način može komunicirati s ostalim računalima na Internetu. Kako je pamćenje ovakvih IP adresa komplicirano postoje i tzv. simboličke IP adrese koje su lakše za pamćenje. Takva simbolička IP adresa poznata je još i kao **host name**. Tako npr. računalo čija je IP adresa 195.29.220.4 ima host name *shk.skolskaknjiga.hr*. Vezivanje IP adrese s host nameom omogućava nam tzv. DNS (Domain Name Server).

Računala spojena na Internet komuniciraju putem jednog od dva protokola:

- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol

Grafički bi to izgledalo ovako:



Zahvaljujući specijalnom Javinom paketu – `java.net` pišući svoje mrežne aplikacije nećemo se trebati zamarati TCP-em, UDP-om, IP-em,... već ćemo ih pisati na aplikacijskom sloju i koristiti sve njegove pogodnosti. Programiranje mrežnih programa na aplikacijskom nivou čini mrežno programiranje poprilično jednostavnim.

U nastavku ćemo se ukratko osvrnuti na najvažnije pojmove vezane uz mrežu koje ćemo susretati u ovom poglavlju:

- TCP
- UDP
- portovi

## TCP protokol

Kada dva računala žele međusobno komunicirati pouzdano, oni će uspostaviti vezu te međusobno slati i primiti podatke putem uspostavljene veze. Ovaj način komunikacije možemo usporediti s slanjem preporučenog pisma. Naime šaljemo li preporučeno pismo tada osoba kojoj pismo šaljemo svojim potpisom potvrđuje da je pismo primila. Ukoliko pismo nije primljeno ono se vraća pošiljatelju. Isto tako TCP nam osigurava da sve što jedno računalo pošalje drugo će sigurno primiti i obrnuto, inače će biti poslana poruka o grešci.

TCP omogućava sigurnu komunikaciju za aplikacije koje to zahtijevaju. HTTP (protokol za prijenos Web stranica), FTP (protokol za prijenos datoteka), i Telnet (protokol za udaljeni rad na računalu) zahtijevaju upravo takvu, pouzdanu komunikaciju. Informacija jesu li podaci stigli na odredište ključna je za ove protokole. Ukoliko ne bi bilo tako primili bismo Web stranicu na kojoj bi nedostajali elementi, skinuli bismo nepotpun dokument,...

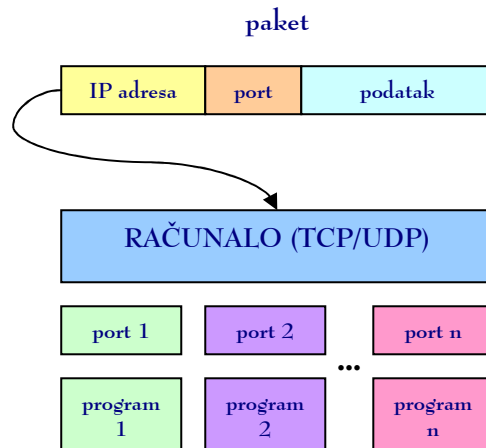
## UDP protokol

Za razliku od TCP protokola koji zahtjeva pouzdanu komunikaciju, UDP je protokol za komunikaciju koji ne garantira da će podaci stići na odredište. Ovaj protokol možemo usporediti s klasičnom poštom. Naime, pošaljemo li obično pismo, nitko nam ne garantira da će ga osoba kojoj je poslano primiti. UDP šalje neovisne pakete podataka tzv. **datagrame** s jednog računala na drugo.

Većina mrežnih aplikacija zahtijevaju pouzdanu komunikaciju, međutim postoje i aplikacije kod kojih to nije bitno.

## Portovi

Već smo rekli da jednom kada se računalo spoji na Internet ono dobiva jedinstvenu IP adresu pomoću koje komunicira s ostalim računalima na Internetu. Svi podaci koje dobiva s Interneta dolaze tom vezom. Međutim računalo može istovremeno primiti podatke za različite aplikacije (downloadati dokument putem FTP-a, učitavati Web stranicu,...). Postavlja se pitanje kako ono zna kojoj aplikaciji treba proslijediti koji podatak? Odgovor leži u portovima. Podaci koji putuju Internetom osim što sadrže IP adresu odredišta sadrže i port na odredištu na koji se trebaju isporučiti.



Na računalu postoji 65536 portova koji su numerirani brojevima od 0 do 65535. Portovi od 0 do 1023 su rezervirani za posebne servise. Tako npr. HTTP koristi port 80, FTP koristi portove 20 i 21, Telnet koristi port 23,... Tablica servisa, protokola te pripadnih portova dana je u dodatku. Mrežne aplikacije koje ćemo mi kreirati također će komunicirati putem portova, rezervirane portove nećemo smjeti koristiti za komunikaciju u svojim aplikacijama.

Internet se temelji na tzv. klijent-server koncepciji. To znači da na Internetu postoje dva osnovna tipa računala:

- klijenti
- serveri (poslužitelji)

**Klijent** je računalno koje serveru šalje zahtjeve i od njega dobiva tražene podatke. **Poslužitelj** je računalno koje sadrži podatke i u stanju je posluživati više računala podacima koje oni od njega traže.

---

## Mrežna komunikacija u Javi

Već smo rekli da nam mrežnu komunikaciju u Javi omogućava paket `java.net`. Paket `java.net` sadrži klase za TCP i UDP komunikaciju.

Najčešće korištene klase za TCP komunikaciju su:

- **InetAddress**
- **URL**
- **URLConnection**
- **Socket**
- **ServerSocket**

Dok su za UDP komunikaciju najčešće korištene klase:

- **DatagramPacket**
- **DatagramSocket**
- **MulticastSocket**

### Klasa InetAddress

Klasa **InetAddress** predstavlja Internet adresu. Najčešće korištene metode ove klase su:

```
static InetAddress getByName (String s) – vraća objekt tipa InetAddress na osnovu imena hosta s  
static InetAddress getByAddress (byte[] a) – vraća objekt tipa InetAddress koji je kreiran na osnovu niza a koji predstavlja IP adresu  
public String getHostAddress () – vraća IP adresu objekta nad kojim se poziva  
public String getHostName () – vraća host name objekta nad kojim se poziva
```

#### Primjer 16 – 1:

Napišimo konzolnu aplikaciju koja će biti pozivana s jednim parametrom, Web adresom neke Web stranice. Aplikacija treba ispisivati IP adresu i host name računala na kojem se Web stranica nalazi.

#### Rješenje:

```
import java.net.*;  
public class IP  
{  
    public static void main (String[] s) {  
        try  
        {  
            InetAddress address = InetAddress.getByName(s[0]);  
            System.out.println(address.getHostName() );  
            System.out.println(address.getHostAddress() );  
        }  
        catch (Exception e)  
        {  
            System.out.println( "Ne mogu pronaci " + s[0]);  
        }  
    }  
}
```

#### Napomena:

Pri izvršavanju programa trebate biti spojeni na Internet.

## Klasa URL

URL (*Uniform Resource Locator*) je jedinstvena adresa bilo kojeg dokumenta na Internetu. URL je tekst koji npr. upisujemo u Address bar Web preglednika kako bi učitali neku stranicu.

Primjer URL-a je:

`http://s5.pro-futura.com:8080/readmail.html`

URL se u osnovi sastoji od nekoliko dijelova:

- **protokol** – neki od Internet protokola, najčešće *http*, *https*, *ftp*,...
- **host name** – naziv računala na kojem je smješten sadržaj
- **port** – broj porta na koji se spajamo (najčešće se ne navodi)
- **ime dokumenta** – put do dokumenta na serveru
- **reference** – mjesto unutar dokumenta na koje će se premjestiti fokus prilikom učitavanja dokumenta

Javina klasa **URL** sadrži metode koje nam omogućuju pristup dokumentima na Internetu.

Klasa **URL** ima nekoliko konstruktora, neki od njih su:

**URL (String s)** – kreira objekt tipa URL pri čemu je s neki URL

**URL (String protokol, String host, int port, String file)** – kreira objekt URL s zadanim parametrima

**URL (String protokol, String host, String file)** – kreira objekt URL s zadanim svojstvima

Neke od metoda definirane nad klasom **URL** su:

**String getFile ()** – vraća file URL-a

**String getHost ()** – vraća host name URL-a

**int getPort ()** – vraća port URL-a. Ako port nije specificiran vraća -1

**String getProtocol ()** – vraća protokol

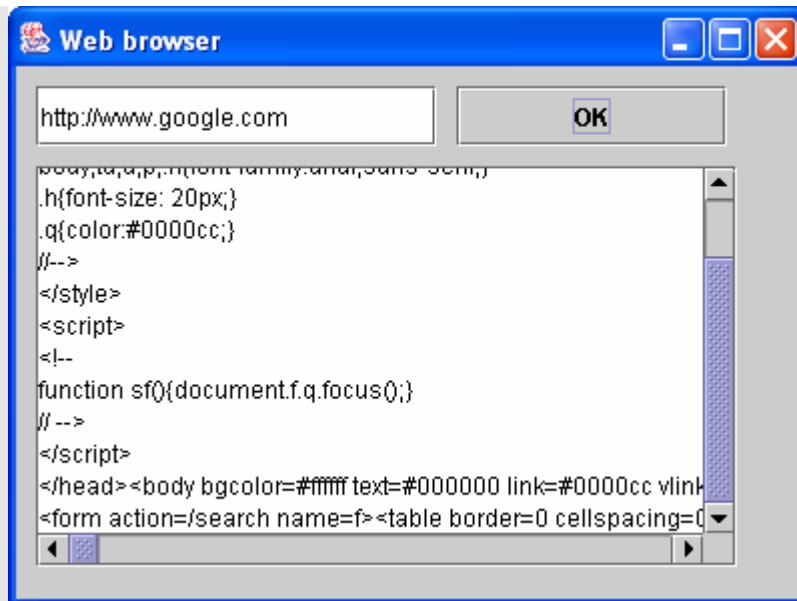
**InputStream openStream ()** – otvara konekciju prema danom URL-u i vraća **InputStream** koji će omogućiti čitanje podataka s URL-a

**URLConnection openConnection ()** – vraća objekt tipa **URLConnection** koji predstavlja konekciju prema URL-u.

### Primjer 16 – 2:

Napišimo program s grafičkim sučeljem koji će unositi URL neke Web stranice i ispisivati sadržaj te stranice u posebnom području za tekst.





### Rješenje:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class WebBrowser extends JFrame implements ActionListener
{
    private final int sirina = 400;
    private final int visina = 300;
    private Container c;
    private JTextArea ta;
    private JTextField t;
    private JScrollPane sp;
    private JButton b;

    public WebBrowser ()
    {
        setTitle ("Web browser");
        setSize (sirina, visina);
        setDefaultCloseOperation (EXIT_ON_CLOSE);

        c = getContentPane ();
        c.setLayout (null);

        t = new JTextField ();
        t.setSize (200, 30);
        t.setLocation (10, 10);
        c.add (t);

        b = new JButton ();
        b.setSize (135, 30);
        b.setText ("OK");
        b.setLocation (220, 10);
        b.addActionListener (this);
        c.add (b);

        ta = new JTextArea ();

        sp = new JScrollPane (ta);
    }
}
```

```
        sp.setLocation (10, 50);
        sp.setSize (350, 200);
        c.add (sp);

        setVisible (true);
    }

    public void actionPerformed (ActionEvent e)
    {
        try
        {
            URL url = new URL(t.getText());
            BufferedReader br = new BufferedReader (new InputStreamReader
            (url.openStream ()));
            String inputLine, tmp = "";
            while ((inputLine = br.readLine()) != null)
                tmp = tmp + "\n" + inputLine;
            ta.setText (tmp);
            br.close();
        }
        catch (Exception e1)
        {
            System.out.println (e1.getMessage());
        }
    }

    public static void main (String[] s)
    {
        WebBrowser z = new WebBrowser ();
    }
}
```

## Klasa **URLConnection**

Klasa **URLConnection** omogućava komunikaciju između programa i URL-a. Metode ove klase omogućavaju čitanje i pisanje na resurs s danim URL-om. Konstruktor ove klase je oblika:

**URLConnection (URL s)** – kreira konekciju za objekt s tipa URL

Nekoliko češće upotrebljivanih metoda klase **URLConnection** su:

**InputStream getInputStream ()** – vraća InputStream za čitanje s otvorene konekcije  
**OutputStream getOutputStream ()** – vraća OutputStream za pisanje na otvorenu URL konekciju

---

## Komunikacija putem socketa

Klase `URL` i `URLConnection` su gotove klase koje omogućavaju komunikaciju sa serverom. U nastavku nam je cilj naučiti kreirati aplikacije koje će komunicirati na nižoj razini. Naučit ćemo kako komunicirati na sloju samog prijenosa podataka, na razini TCP odnosno UDP protokola.

TCP omogućava tzv. point-to-point komunikaciju između klijenta i servera. Za komunikaciju putem TCP-a serverske i klijentske aplikacije koriste poseban komunikacijski kanal kojim prolaze podaci. Klijent je u stvari računalo na kojem se izvršava klijentska aplikacija dok s druge strane imamo server tj. računalo na kojem se izvršava serverska aplikacija. Kanal između servera i klijenta je dvosmjernan što znači da njime putuju podaci od klijenta prema serveru i od servera prema klijentu. Krajeve takvog kanala nazivamo **socketi**. Na komunikacijski kanal možemo gledati kao na hodnik koji povezuje dvije prostorije, dok bi socketi u tom slučaju bila vrata na svakom od krajeva hodnika.

Dakle, kada se uspostavi komunikacijski kanal između dva računala na svakom kraju tog kanala otvara se po jedan socket.

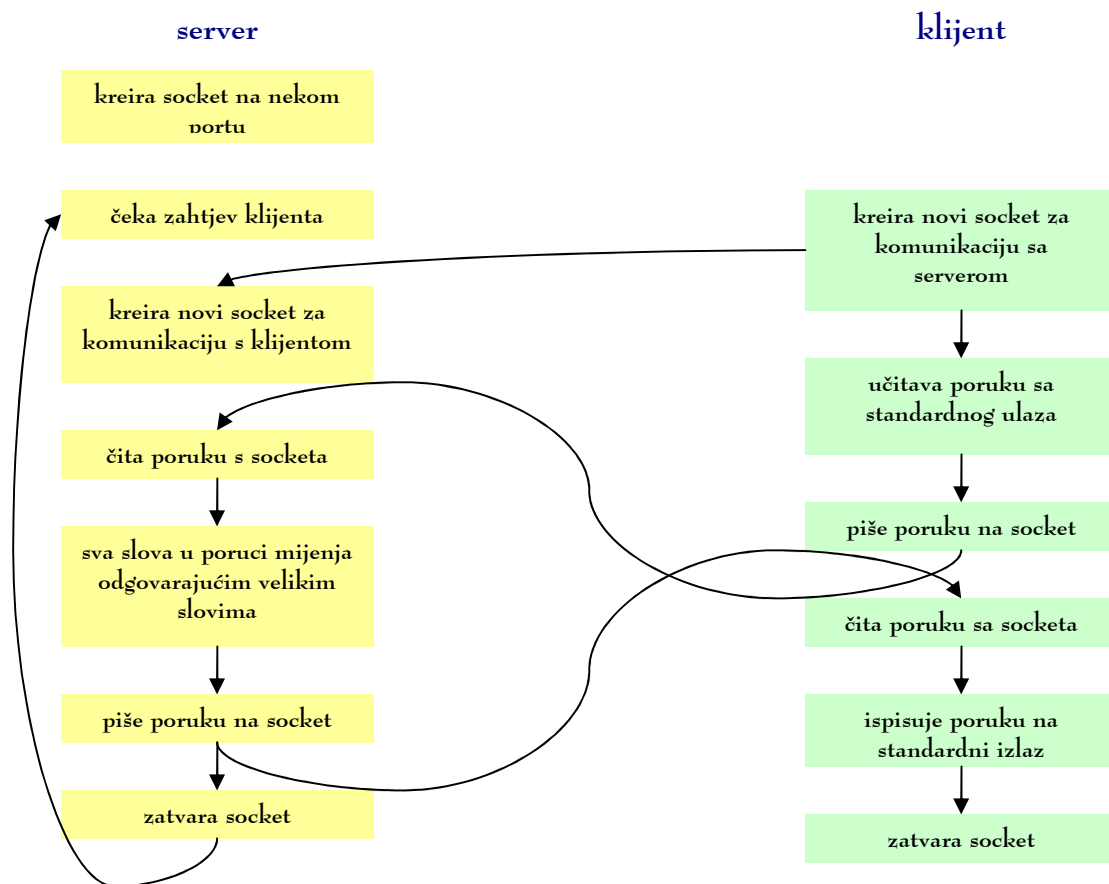
S jedne strane imamo server koji je samostalna aplikacija na nekom računalu. Server ima socket na nekom portu i čeka da klijent pošalje zahtjev za početkom komunikacije.

S druge strane klijent kreira svoj socket, on zna IP adresu računala na kojem se nalazi server te zna port na kojem je server. Kada klijent kreira socket serveru se šalje zahtjev za uspostavom veze. Kada primi zahtjev od klijenta server kreira novi socket, na novom portu, za komunikaciju s klijentom, te je na taj način omogućena komunikacija s više klijenata istovremeno.

Nakon što je komunikacijski kanal otvoren moguće je prenositi podatke između servera i klijenta. Za početak nam je želja kreirati jednostavnu aplikaciju koja će raditi na sljedećem principu:

- klijent učitava liniju teksta sa standardnog ulaza te ju ispiše na socketu.
- server sa socketa pročita tekst koji je klijent poslao, sva slova poruke pretvori u velika te takvu, izmijenjenu poruku ispiše na socket
- klijent pročita poruku sa socketa te ju ispiše ponovo na standardni izlaz

Shematski prikaz opisanih radnji mogao bi izgledati ovako:



Da bismo ovo bili u stanju isprogramirati moramo znati nešto o socketima u Javi. Kako pročitati podatke sa socketa? Kako pisati na socket?...

Javina klasa `java.net` sadrži klasu **Socket** koja implementira jedan kraj komunikacijskog kanala.

Isto tako `java.net` sadrži i klasu **ServerSocket** koja implementira socket koji će koristiti server za čekanje i prihvatanje konekcije od strane klijenta.

Klasa **Socket** ima nekoliko konstruktora najčešći su:

**Socket (String host, int port)** – kreira socket sa specificiranim hostom i portom  
**Socket (InetAddress address, int port)** – kreira socket sa specificiranom adresom i portom

Najčešće metode definirane nad klasom `Socket` su:

**public void close ()** – zatvara konekciju  
**public InetAddress getInetAddress ()** – vraća adresu s kojom je klijent spojen  
**public InputStream getInputStream ()** – kreirao objekt tipa `InputStream` koji će nam omogućavati čitanje podataka sa socketa  
**public InetAddress getLocalAddress ()** – vraća lokalnu adresu računala na kojem je socket  
**public int getLocalPort ()** – vraća lokalni port na kojemu je socket

**public OutputStream getOutputStream ()** – kreira objekt tipa **OutputStream** koji će nam omogućavati pisanje podataka na socket

Najčešći konstruktor klase **ServerSocket** je:

**ServerSocket (int port)** – kreira objekt tipa **ServerSocket** na specificiranom portu

Najčešće korištene metode ove klase su:

**public Socket accept ()** – očekuje zahtjev klijenta i kreira novi socket

**public void close ()** – zatvara socket

**public InetAddress getLocalAddress ()** – vraća lokalnu adresu računala na kojem je socket

**public int getLocalPort ()** – vraća port na kojemu je socket čeka

Sada imamo sve spremno za rješavanje problema koji smo si postavili.

### Primjer 16 – 3:

Kreirajmo server aplikaciju na portu 2000 koja će od klijenta uzimati poruku te sva slova poruke pretvoriti u velika. Tako dobivenu poruku treba ispisati na socket.

Potrebno je kreirati i klijent aplikaciju koja će se spajati sa serverom na port 2000, učitavat će poruku sa standardnog ulaza, pisati ju na socket te sa socketa uzeti poruku koju je vratio server i ispisati ju na standardni izlaz.

### Rješenje:

Klasa za serversku aplikaciju

```
import java.net.*;
import java.io.*;
public class Server
{
    private ServerSocket ss;
    private int port;
    private Socket s;
    public Server(int port)
    {
        this.port = port;
        try
        {
            //otvaranje socketa na serveru na portu port koji će čekati
            //zahtjev klijenta za otvaranje komunikacijskog kanala
            ss = new ServerSocket (port);
            System.out.println ("Socket kreiran");

            //čeka se zahtjev klijenta za spajanje na server
            s = ss.accept ();

            //zaprmljen zahtjev klijenta i otvoren je komunikacijski kanal
            //na serveru je kreiran novi socket
            System.out.println ("Klijent spojen");

            //objekt za čitanje podataka sa socketa s
            InputStream in = s.getInputStream();

            //objekt za pisanje podataka na socket s
```

```
        OutputStream out = s.getOutputStream();

        //čitanje podataka sa socketa - rečenica koju je na socket
        //zapisao klijent
        BufferedReader br = new BufferedReader (new InputStreamReader
        (in));
        String tmp = br.readLine();

        System.out.println ("Primio sam poruku od klijenta: " + tmp);
        //pretvaranje svih slova u velika
        tmp = tmp.toUpperCase ();

        System.out.println ("Saljem poruku klijentu: " + tmp);

        //pisanje na socket
        PrintStream ps = new PrintStream (out);
        ps.println (tmp);

        //zatvaranje konekcije
        s.close();
    }
    catch (Exception e)
    {
        System.out.println ("Greska");
    }
}

public static void main (String[] s)
{
    Server sr = new Server (2000);
}
}
```

Klasa za klijentsku aplikaciju

```
import java.net.*;
import java.io.*;
public class Klijent
{
    private int port;
    private String host;
    private Socket s;
    public Klijent(String host, int port)
    {
        try
        {
            this.host = host;
            this.port = port;

            //otvaranje socketa prema serveru na portu 2000
            s = new Socket (host, port);

            //Čitanje podataka sa standardnog ulaza
            BufferedReader br = new BufferedReader (new InputStreamReader
            (System.in));
            System.out.println ("Unesi poruku: ");
            String tmp = br.readLine ();

            //pisanje poruke na socket
            OutputStream out = s.getOutputStream();
            PrintStream ps = new PrintStream(out);
```

```

        ps.println (tmp);

        //čitanje poruke sa socketa
        InputStream in = s.getInputStream();
        BufferedReader bs = new BufferedReader (new InputStreamReader
        (in));
        String tmp2 = bs.readLine();

        //ispis poruke na standardni output (ekran)
        System.out.println (tmp2);
    }
    catch (Exception e)
    {
        System.out.println ("Greska");
    }
}

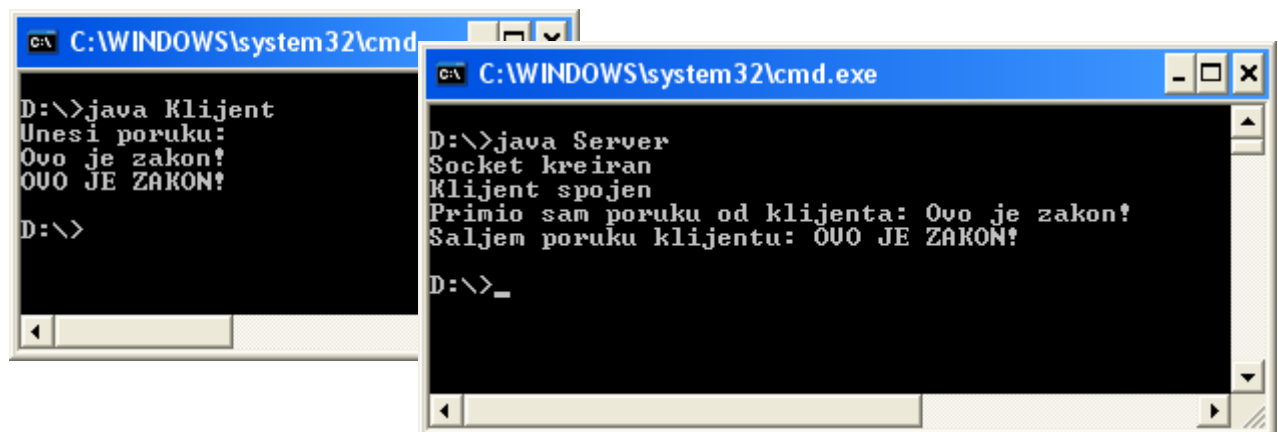
public static void main (String[] s)
{
    //server na koji se spajamo u ovom se slučaju zove predrag
    Klijent k = new Klijent ("localhost", 2000);
}
}

```

### Napomena:

Program ćemo testirati tako da na jednom računalu prvo pokrenemo klasu **Server**. Ukoliko imamo dva računala koja su umrežena na drugom računalu ćemo pokrenuti klasu **Klijent** (klase ćemo pokrenuti iz komandnog prompta). Ukoliko imamo samo jedno računalo, možemo obje klase pokrenuti na istom računalu.

Ukoliko obje klase pokrećemo na istom računalu kao host name kod klijenta možemo koristiti *localhost*.



## Višekorisnički server

U prethodnom primjeru smo kreirali klijent-server aplikaciju kod koje smo imali jedan server i samo jednog klijenta, koji su izmijenili samo jednu poruku i komunikacija se je prekinula. Na samom početku smo rekli da nam je cilj kreirati vlastiti chat. Kao što znamo na chat se može spojiti više korisnika (klijenata) koji međusobno razgovaraju. Svaki klijent šalje poruku koju vide svi sudionici chata.

Dakle, želimo li napraviti chat, moramo kreirati server koji će biti u stanju prihvatiti više klijenata i s njima komunicirati. Server bi trebao biti u stanju u svakom trenutku biti u stanju primiti novog klijenta i svi klijenti bi trebali biti međusobno neovisni. Dakle, na serveru bi trebala postojati jedna petlja koja će neprestano čekati i prihvaćati nove konekcije.

Dio servera koji će biti u stanju neprekidno prihvaćati nove konekcije mogao bi izgledati ovako:

```
ss = new ServerSocket( port );
System.out.println( "Cekam klijente.... " );
while (true)
{
    Socket s = ss.accept();
    //nakon što se klijent spoji na neki port s njim nešto napravim
    System.out.println( "Klijent spojen na port " + s.getPort() );
}
```

Na ovaj smo način omogućili istovremeno spajanje više klijenata na server. Međutim nakon što se klijent spoji na server, on ništa ne radi, samo se na serveru ispiše poruka da je klijent spojen i port na kojem je spojen. Primijetimo da se svi klijenti spajaju na socket **s** na serveru. Dakle kada se spoji novi klijent u principu gubimo svaki odnos sa starim klijentom.

U stvari bi bilo idealno kada bi naš klijent mogao biti nešto kao recepcionar u nekom bogatom hotelu. Dakle, server bi trebao "prihvatiti" klijenta, "zapisati ga" i zatim ga "prepustiti pomoćnom osoblju" koje će mu "stajati na usluzi", te ponovno "čekati" novog klijenta...

Dakle, rezimirajmo: naš server će prihvaćati klijente te ih prosljeđivati "nekom drugom" koji će dalje voditi računa o svakom klijentu. Nakon što proslijedi klijenta "nekome drugom" on će biti ponovo slobodan i čekati će nove klijente. Taj "netko drugi" će biti posebna klasa koja će prihvaćati klijenta i s njim nešto raditi. Klasa će evidentirati neke osnovne podatke o klijentu (ime, socket na koji je spojen,...) te će u sebi imati implementiranu komunikaciju klijenta sa serverom. Nakon što se klijent "ugasi" ova će klasa poslati poruku serveru da se klijent više nije spojen.

Na ovaj smo način uistinu osigurali da server uistinu ne treba više voditi računa o klijentu, nakon što ga prihvati. Još nam je preostao samo jedan problem, kako to sve realizirati u Javi?



## Višenitnost

Snaga Jave kao programskog jezika najviše dolazi do izražaja u mrežnim i tzv. višenitnim (multithreading) aplikacijama.

U nastavku ćemo pokušati objasniti kakve su to višenitne aplikacije. Pokušajmo to objasniti na jednom jednostavnom primjeru: taksisti služba unutar nekog grada raspolaže s nekoliko taksista koji stoje na raspolaganju. Na poziv klijenta koji treba prijevoz taksist dođe po njega te mu stoji na usluzi i vozi ga gdje god klijent želi, naravno uz određenu naknadu. Svi taksisti rade unutar jedne taksisti službe ali svaki od njih razvozi klijente neovisno kada se za to pojavi potreba. Nerijetko se događa da više taksista razvozi klijente istovremeno.

Slična je situacija i kod višenitosti unutar programa. Dakle, imamo jedan program ali unutar kojega se istovremeno odvija više neovisnih procesa (niti, threads). Svaka nit vodi računa o izvršavanju određenog skupa naredbi. Višenitni programi imaju višestruke, autonomne aktivnosti koje se izvršavaju istovremeno.

To je upravo ono što nam treba u našem chatu. Server može istovremeno komunicirati s više klijenata, međutim klijenti su znatno sporiji od servera. Stoga će server glavninu vremena čekati da neki klijent nešto kaže. Ako imamo jednonitnu aplikaciju koja čeka da klijent 1 nešto kaže, u to vrijeme bi možda neki drugi klijenti možda već izmijenili nekoliko poruka, jer npr. klijent 1 sporo piše.

Stoga ćemo kreirati višenitnu aplikaciju kod koje će postojati niti za svakog klijenta koji se spajina server. Višenitnost će nam omogućiti da klijenti šalju poruke serveru neovisno, a znamo da je server dovoljno brz i on će sve te poruke biti u stanju obrađivati.

Za rad s nitima Java ima posebnu klasu **Thread**. Klasa **Thread** implementira interface **Runnable** koja ima jednu public metodu: **run ()**, koja ništa ne vraća. Ukratko to znači da ćemo u klasi koja će u sebi koristiti klasu **Thread** trebati definirati metodu **run** sa zaglavljem: **public void run ()**.

Unutar tijela metode **run** definirat ćemo "ponašanje" klase. Tj. definirat ćemo što će se događati s konkretnom niti unutar nekog programa.

Kao i svaka druga klasa i klasa **Thread** ima nekoliko konstruktora:

**Thread ()** – kreira novi objekt tipa **Thread**

**Thread (String name)** – kreira novi objekt tipa **Thread** kojemu će biti dodijeljeno dano ime (*name*)

Neke od metoda definirane unutar klase **Thread** su:

**static int activeCount ()** – vraća broj trenutno aktivnih niti

**void destroy ()** – uništava nit

**String getName ()** – vraća ime niti

**void start ()** – pokreće nit, tj. započinje izvršavanje metode **run ()**

Sada imamo sve spremno za kreiranje klase koja će brinuti o svakom klijentu. Klasa će kao svojstva imati ime serverske aplikacije koja je kreirala vezu s klijentom i koja je pokrenula nit programa. Osim toga znat će ime socketa preko kojega se odvija komunikacija te će imati informaciju o nekom simboličkom imenu klijenta koje mu je dodijelio server kako bi se znalo točno znalo koji klijent šalje informacije.

Zadaća niti će biti da preuzme poruku sa ulaza te ju stavi na socket i kaže serveru da poruku pročitati i pošalje ju svim klijentima koji su spojeni na chat.

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

```

public class ChatServerNit extends Thread
{
    private ChatServer server;
    private Socket socket;
    private String name;

    public ChatServerNit ( ChatServer server, Socket socket, String name )
    {
        //podaci o serveru, socketu i simboličko ime klijenta su svojstva
        //klase ChatServerNit
        this.server = server;
        this.socket = socket;
        this.name = name;

        //početak "rada" niti
        start();
    }

    //metoda run koja opisuje rad klijenta
    public void run()
    {
        try
        {
            BufferedReader din = new BufferedReader( new InputStreamReader
            ( socket.getInputStream() ) );

            //beskonačno ponavlja
            while (true)
            {
                //Učitaj poruku s ulaza
                String message = din.readLine ();

                //ispiši poruku na socket
                System.out.println(message );

                //upozoravamo server da je klijent poslao poruku, tako
                //da pozovemo odgovarajuću metodu na serveru.
                server.sendToAll( name + " > " + message );
            }
        }
        catch( Exception ie )
        {
            System.out.println(ie.getMessage());
        }

        //nakon što se klijent "ugasi" javlja se poruka serveru da
        //obriše klijenta
        server.removeConnection( socket );
    }
}

```

Preostaje nam još izmijeniti server. Server će voditi evidenciju o svim klijentima spojenim na server i to pomoću posebnog niza socketa. Isto tako će za svaki socket imati posebni `PrintStream` koji će mu omogućiti da zna na koje sve sockete treba pisati poruke:

```

import java.io.*;
import java.net.*;
public class ChatServer
{
    private ServerSocket ss;

```

```

//na server će se moći spojiti maksimalno 100 klijanata
private final int MAX = 100;

//podaci o socketima pamtit će se unutar niza soc
private Socket[] soc = new Socket[MAX];

//za svaki socket bit će kreiran poseban PrintStream
private PrintStream[] ps = new PrintStream[MAX];
private int elements = 0, k = 1;
public ChatServer( int port )
{
    try
    {
        ss = new ServerSocket( port );
        System.out.println( "Cekam klijente.... " );
        while (true)
        {
            //prihvatanje klijenta i otvaranje novog socketa koji
            //će se spremati u niz socketa te kreiranje
            //posebnog PrintStreamova za svaki socket
            Socket s = ss.accept();
            System.out.println("Klijent spojen na port " + s.getPort());
            PrintStream dout = new PrintStream(s.getOutputStream());
            soc [elements] = s;
            ps [elements++] = dout;

            //kreiranje niti za novog klijenta, imena klijenata bit će
            //redni brojevi njihovog spajanja na server
            sendToAll ( "Dobrodosao Klijent " + k );
            new ChatServerNit( this, s, "Klijent " + k++ );
        }
    }
    catch (Exception e)
    {
        System.out.println (e.getMessage ());
    }
}

//metoda koja pristiglu poruku piše na sve sockete
void sendToAll( String message )
{
    for (int i = 0; i < elements; i++)
    {
        PrintStream dout = ps [i];
        try
        {
            dout.println (message);
        }
        catch( Exception e )
        {
            System.out.println( e.getMessage ( ) );
        }
    }
}

//briše klijenta koji je otišao s chata, tako da briše odgovarajuće
//elemente iz niza socketa i niza PrintStreamova
void removeConnection( Socket s )
{
    System.out.println( "Brišem klijenta na portu " + s.getPort() );
    int i = 0;

```

```
        while (!soc [i].equals (s))
            i++;

        //sve elemente iza izbrisanog pomiče za jedno mjesto unatrag
        for (int j = i; j < elements; j++)
        {
            soc [j] = soc [j + 1];
            ps [j] = ps [j + 1];
        }
        elements--;
        System.out.println( name + " je napustio chat" );
        sendToAll ( name + " je napustio chat");
    }

    public static void main( String args[] )
    {
        try
        {
            int port = Integer.parseInt( args[0] );
            ChatServer cs = new ChatServer( port );
        }
        catch (Exception e)
        {
            System.println (e.getMessage());
        }
    }
}
```

Preostaje nam još kreirati klijenta. Klijent ćemo poput pravog chat klijenta implementirati kao Applet te ga staviti unutar neke Web stranice.

Chat klijent treba voditi računa o dvije stvari:

- nakon što korisnik upiše poruku chat klijent ju treba zapisati na socket
- neprekidno provjeravati ima li novih poruka na socketu te ukoliko ih ima ispisati ih u svom prozoru

Prvi problem: slanje upisane poruke na socket riješit ćemo prilično jednostavno. Korisnik chata će upisati poruku u poseban tekstualni okvir te će pritiskom na tipku [enter] zapisati na socket. To ćemo napraviti tako da ćemo kreirati Action Listener na tekstualnom okviru. Pritiskom tipke [enter] kreirat će se događaj koji će pokrenuti metodu **sendMessage ( )**, koja će poruku zapisati na socket. Metoda **sendMessage ( )** vrlo je jednostavna i ne predstavlja nikakvu novost, ona izgleda ovako:

```
private void sendMessage( String message )
{
    try
    {
        dout.println ( message );
        tf.setText( "" );
    }
    catch( Exception e )
    {
        System.out.println( e.getMessage() );
    }
}
```

pri čemu je *dout* objekt tipa `PrintStream`.

Naš chat klijent osim što piše podatke na socket te ih na taj način prosljeđuje serveru, on mora i provjeravati je li server upisao kakve podatke na socket, a ako ih je upisao treba ih pročitati i ispisati unutar sučelja chat klijenta.

Slična kao i kod servera i ovdje ćemo morati kreirati posebnu nit koja će u pozadini izvršavati beskonačnu petlju koja će neprestano provjeravati ima li podataka na socketu te ih pročitati i ispisati u prozoru chat klijenta. To ćemo implementirati unutar metode `run()`. Prisjetimo se svaki puta kada koristimo klasu `Thread`, unutar svoje klase trebamo definirati metodu `run()`.

Ta je metoda poprilično jednostavna i ima sljedeći oblik:

```
public void run()
{
    try
    {
        while (true)
        {
            String message = din.readLine();
            ta.append( message + "\n" );
        }
    }
    catch( Exception e )
    {
        System.out.println( e.getMessage() );
    }
}
```

Dakle, cjelokupna implementacija chat klijenta bila bi:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
public class ChatClient extends Applet implements Runnable, ActionListener
{
    private TextField tf = new TextField();
    private TextArea ta = new TextArea();
    private Socket socket;
    private PrintStream dout;
    private BufferedReader din;

    public void init()
    {
        ta.setEditable (false);
        String host = getParameter( "host" );
        int port = Integer.parseInt( getParameter( "port" ) );
        setLayout( new BorderLayout() );
        add("Center", ta);
        add("South", tf);
        tf.addActionListener(this);
        try
        {
            Thread t = new Thread (this);
            socket = new Socket( host, port );
            din = new BufferedReader( new InputStreamReader
            (socket.getInputStream()) );
            dout = new PrintStream( socket.getOutputStream() );
            t.start();
        }
    }
}
```

```
        catch( Exception e )
        {
            System.out.println( e.getMessage() );
        }
    }

    public void actionPerformed (ActionEvent e)
    {
        sendMessage (tf.getText ());
    }

    private void sendMessage( String message )
    {
        try
        {
            dout.println ( message );
            tf.setText( "" );
        }
        catch( Exception e )
        {
            System.out.println( e.getMessage() );
        }
    }

    public void run()
    {
        try
        {
            while (true)
            {
                String message = din.readLine();
                ta.append( message + "\n" );
            }
        }
        catch( Exception e )
        {
            System.out.println( e.getMessage() );
        }
    }
}
```

### Zadaci za vježbu:

1. Što je protokol?
2. Što je IP adresa?
3. Koji su osnovni protokoli za komunikaciju putem Interneta i objasni svaki od njih.
4. Što je port?
5. Koliko portova postoji na računalu?
6. Koje portove je moguće koristiti u svojim aplikacijama?
7. Objasni klijent-server princip komunikacije.
8. Nabroji neke klase za mrežnu komunikaciju u Javi.
9. Kreirajte aplikaciju s grafičkim korisničkim sučeljem koja će omogućavati unos web adrese te će klikom na gumb **IP** vraćati odgovarajuću IP adresu.
10. Što nam omogućavaju klase **URL** i **URLConnection**?
11. Napišite applet koji će npr. s URL-a Hrvatske narodne banke (*www.hnb.hr*) uzimati tečaj eura i dolara te ga ispisivati na Web stranici.
12. Što je socket?
13. Kreirajte mrežnu klijent-server aplikaciju koja će omogućiti komunikaciju između klijenta i servera. Klijent učitava poruke sa standardnog ulaza i šalje ih serveru sve dok se s klijenta ne pošalje poruka *KRAJ*. Dakle aplikacija treba raditi sljedeće:
  - klijent učitava poruku,
  - klijent šalje poruku serveru,
  - server preuzima poruku od klijenta,
  - server sva slova poruke pretvori u velika,
  - server pošalje poruku klijentu
  - klijent ispisuje poruku na izlaz
  - klijent učitava poruku sa standardnog ulaza
  - ...
  - Aplikaciju kreirajte s grafičkim korisničkim sučeljem.
14. Kreirajte mrežnu klijent-server aplikaciju koja će biti jednostavan Web server za tekstualne dokumente:
  - klijentska aplikacija treba unositi ime dokumenta na serveru
  - klikom na gumb Učitaj sa servera se treba u poseban okvir učitati sadržaj tekstualnog dokumenta sa servera, odnosno ukoliko dokument ne postoji treba ispisati odgovarajuću poruku
  - server treba obraditi zahtjev klijenta i poslati sadržaj dokumenta klijentu
  - Aplikaciju kreirajte s grafičkim korisničkim sučeljem.
15. Objasni višenitnost (multithreading).
16. Izmijenite chat aplikaciju tako da:
  - a. Svaki korisnik može pri ulazu u chat unijeti svoje ime;
  - b. Chat aplikacija treba kreirati tzv. log file, unutar tog log filea trebaju se nalaziti čitava arhiva o chatu (tko je pristupio chatu, u koje vrijeme, što je pisao,...)
17. Kreirajte mrežnu aplikaciju koja će omogućiti slanje tekstualnih dokumenata s jednog računala na drugo.



# 17

## Osnove baza podataka u Javi

- Što je baza podataka?
- SQL (Structured Query Language)
- Java i baze podataka
- Još neki elementi grafičkog korisničkog sučelja



U profesionalnom programerskom svijetu veliki dio programiranja svodi se na pisanje programa koje u svojoj pozadini imaju baze podataka. Cilj ovog poglavlja je naučiti raditi s bazama podataka te naučiti kako povezivati baze podataka s programima pisanim u Javi.

O bazama podataka napisano je na stotine knjiga. Mi nećemo ulaziti u dubinu baza podataka, reći ćemo nekoliko osnovnih činjenice rada s bazama podataka koje će nam omogućiti izradu jednostavnog adresara ali i niza drugih, sličnih aplikacija.

S bazama podataka ćemo se upoznati kreirajući jednostavan adresar. Adresar bi trebao biti program u kojem će se nalaziti podaci o osobama.

## Specifikacija programa

U adresar će se spremati sljedeći podaci o osobi:

- ime
- prezime
- datum rođenja
- telefon
- mobitel
- e-mail
- adresa

Aplikacija treba imati sljedeće funkcionalnosti:

- dodavanje podataka o novim osobama
- pretraživanje osoba prema svim kriterijima (ime, prezime, datum rođenja, telefon, mobitel, e-mail, adresa) i to tako da omogućava pretraživanje s zamjenskim znakom \*
- brisanje osoba iz adresara
- izmjenu podataka o osobama u adresaru

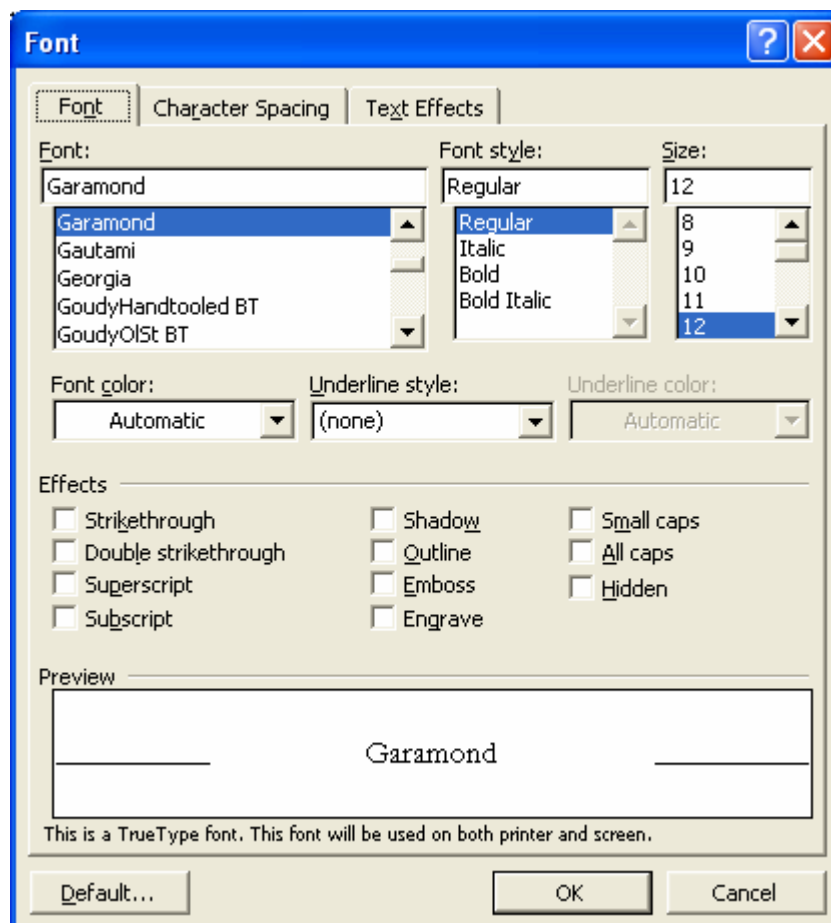
## Izvedba programa

Program će biti napisan kao Java aplikacija. Podaci za program nalazit će se u Accessovoj bazi podataka, a pristupat će im se iz programa pomoću posebnih upita.

Program će biti organiziran kroz dva različita sučelja:

- sučelje za unos podataka
- sučelje za pretraživanje, brisanje i izmjenu podataka

Sučelja će biti implementirana kao tzv. jahači unutar jednog prozora, primjer takvog prozora s jahačima je:



Slika 17-1 Prozor s tzv. jahačima (*Font, Character Spacing, Text Effects*)

U dijelu za unos nalazit će se tekstualna polja za unos: imena, prezimena, datuma rođenja, telefona, mobitela, e-maila, te područje za tekst za unos adrese. Na kraju će se nalaziti i jedan gumb **Spremi**. Klikom na gumb spremi će se u bazu podataka upisati podaci upisani u odgovarajuća polja za unos. Te će se ispisati odgovarajuća poruka ukoliko su podaci uspješno spremljeni u bazu, odnosno poruka o grešci ako je došlo do greške prilikom spremanja podataka u bazu (krivi format datuma,...).

Pretraživanje podataka biti će organizirano tako da će se neki od kriterija pretraživanja upisati u poseban okvir za tekst. Klikom na gumb **Traži** pretražit će se svi podaci u tablici. Ime i prezime svih osoba koje u nekom od svojih podataka (ime, prezime, telefon,...) imaju zadani kriterij bit će ispisani u posebnoj listi na ekranu. Klikom na određeni zapis u listi, podaci o odabranoj osobi bit će prikazani u odgovarajućim poljima te će se moći izmijeniti i spremiti izmjene. Isto tako će se klikom na osobu unutar popisa moći izbrisati ta osoba iz baze podataka.

## Što je baza podataka?

Rekli smo da ćemo podatke o osobama čuvati unutar Accessove baze podataka. U nastavku ćemo reći nešto više o bazama podataka općenito te o kreiranju baze podataka u MS Accessu. Jednostavno rečeno, baza podataka je skup tablica u kojima su zapisani podaci. Svaka tablica sastoji se od stupaca, pri čemu svaki stupac ima svoje ime. Podaci su zapisani u redcima. Npr. tablica s popisom učenika neke škole, unutar neke baze podataka mogla bi izgledati ovako:

Ime	Prezime	Datum rođenja	Spol
Ines	Marković	13.2.1987.	M
Ivana	Bosanac	14.9.1988.	Ž
Iva	Vekić	6.1.1977	Ž
Matko	Kuzmanić	13.9.1980	M

Kod tablica u bazi podataka dobro je uvijek imati stupac takav da podaci unutar njega jedinstveno određuju određeni redak. Pojednostavljeno rečeno. U našoj tablici trebao bi postojati stupac koji bi jedinstveno određivao svaki redak unutar tablice. U našoj tablici takvog stupca očito nema. Naime, *ime* definitivno nije takav stupac jer dvije osobe mogu imati isto ime. Slično je i s *prezimenom*, *datumom rođenja* te *spolom*. Takav stupac koji jedinstveno određuje podatke u retku zovemo **primarni ključ** (*Primary key*). U našoj bi tablici primarni ključ mogao biti npr. JMBG. Kao što znamo, ne može se dogoditi da dvije osobe imaju isti JMBG.

Vrlo često se kao primarni ključ postavljaju brojevi koji se automatski povećavaju dodavanjem novog zapisa u tablicu. Pravi smisao primarnog ključa dobije se kada u bazi imamo više tablica čiji su podaci međusobno povezani, no o tome ćemo govoriti nešto kasnije. Dakle, naša tablica s primarnim ključem, koji se automatski povećava dodavanjem novog zapisa, mogla bi izgledati ovako:

ID	Ime	Prezime	Datum rođenja	Spol
1	Ines	Marković	13.2.1987.	M
2	Ivana	Bosanac	14.9.1988.	Ž
3	Iva	Vekić	6.1.1977	Ž
4	Matko	Kuzmanić	13.9.1980	M

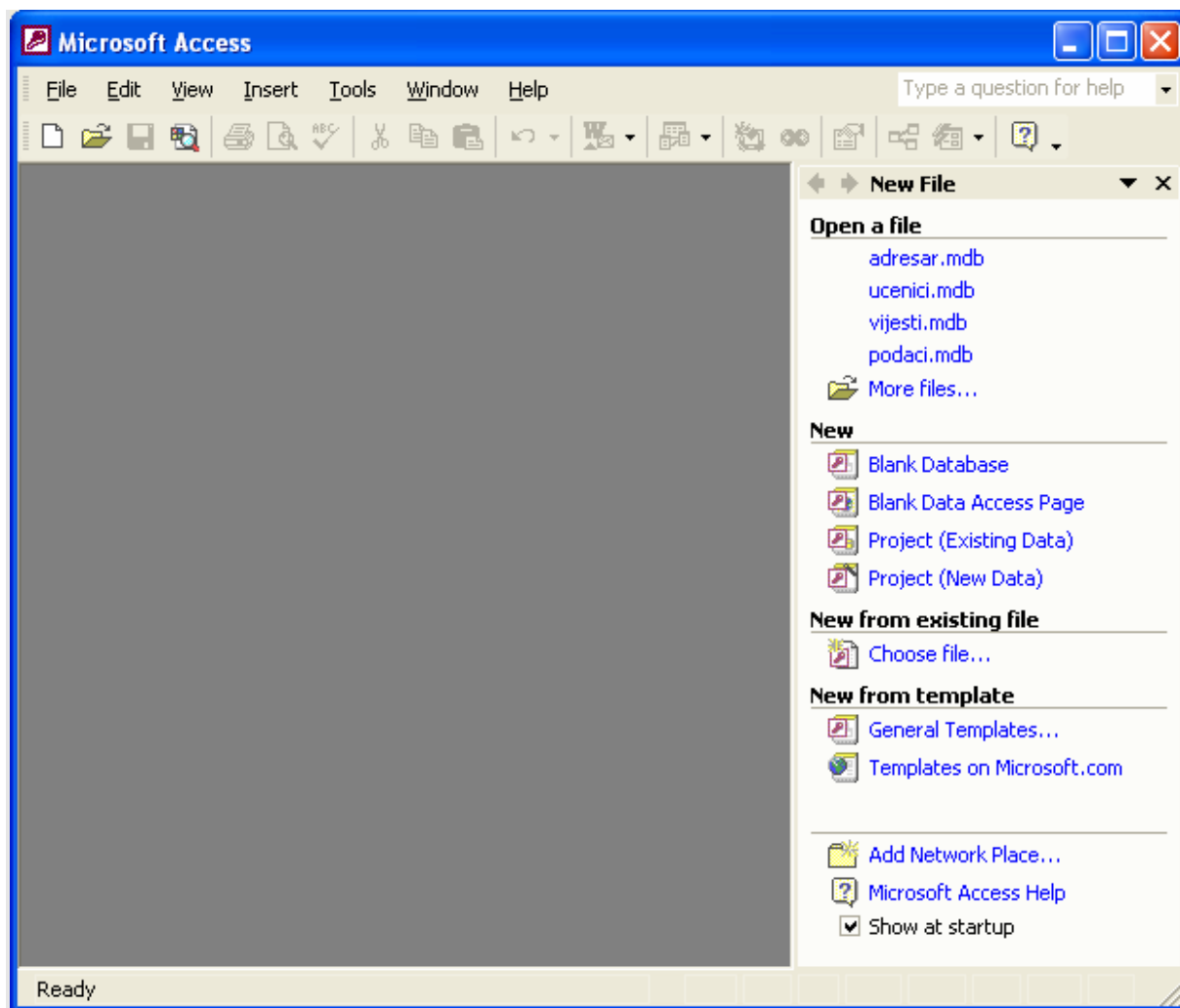
Danas postoji mnoštvo programa za rad s bazama podataka (MS SQL, Oracle, MySQL,...), nama najbliži, najjednostavniji i s daleko najmanjim brojem mogućnosti je MS Access. Radi se o programu koji dolazi u kompletu s MS Officeom. U nastavku ćemo naučiti kako u Accessu kreirati tablicu za naš *Adresar*.

### Primjer 17 – 1:

Kreirajmo tablicu koja će sadržavati sve stupce potrebne za program **Adresar**.

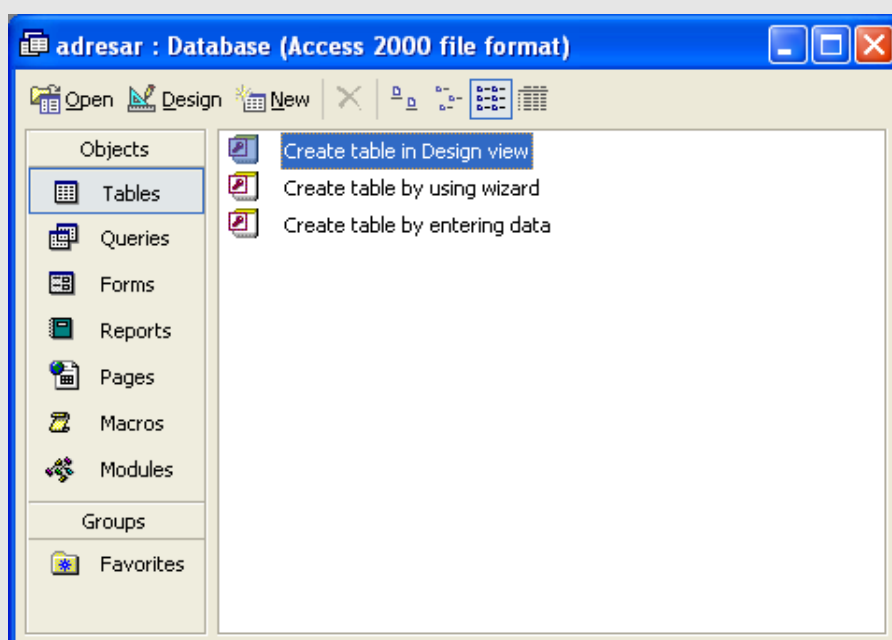
#### Rješenje:

Pokrenimo MS Access. Ovdje ćemo raditi na MS Accessu XP, sve ostale verzije vrlo su slične.



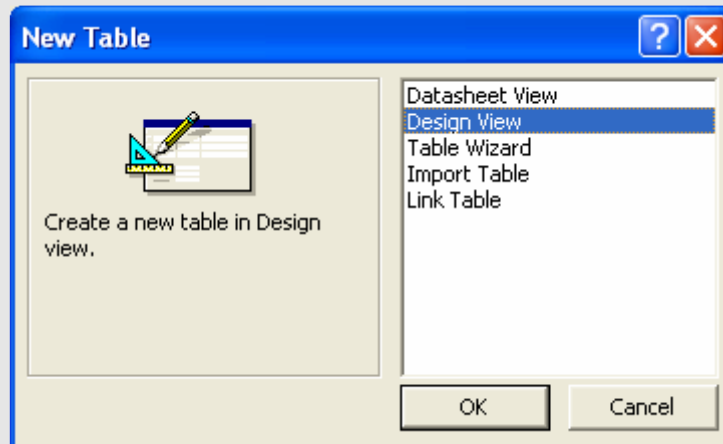
Slika 17-2. Prozor MS Accessa

U desnom dijelu prozora odaberemo **Blank Database**. Odaberemo mapu gdje želimo spremiti bazu te upišemo ime (*adresar.mdb*) i kliknemo na gumb **Save**.



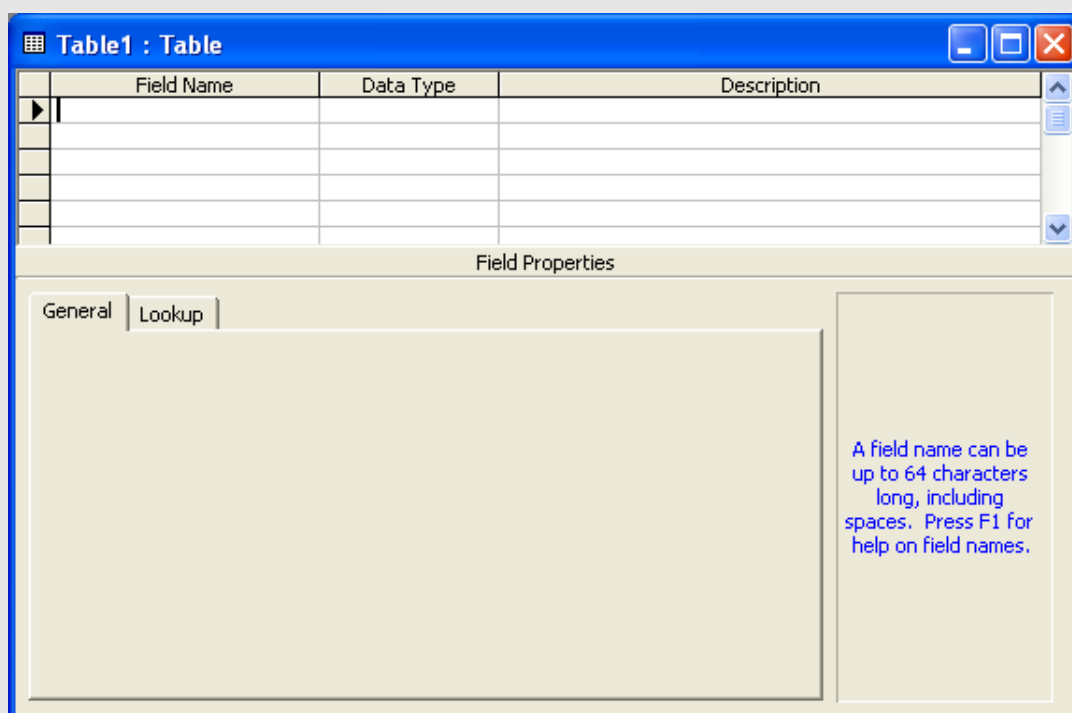
**Slika 17-3.** Glavni prozor za rad s bazom podataka

U prozoru koji se otvori, kliknemo na sličicu **Tables** u lijevom dijelu prozora, te odaberemo **New** u gornjem dijelu prozora.



**Slika 17-4.** Prozor za kreiranje tablice unutar baze podataka

Kliknemo na **Design View** te definirajmo stupce unutar tablice:



**Slika 17-5.** Definiranje stupaca tablice

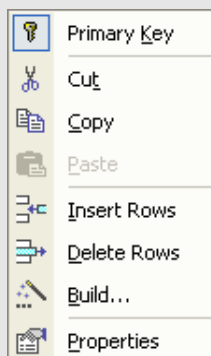
U stupac **Field name** upisivat ćemo nazive stupaca. Dok ćemo pod **Dana Type** odabirati tip vrijednosti koje ćemo upisivati unutar stupaca. Najčešći tipovi vrijednosti je *Text* odnosno *Number* za unos teksta odnosno brojeva. Često se još koriste i *Date* za datum i vrijeme, te *Autonumber* koji označava da se vrijednost stupca automatski povećava za 1.

Mi ćemo kreirati sljedeće stupce:

- **ID** – Autonumber
- **ime** – Text, najviše 30 znakova

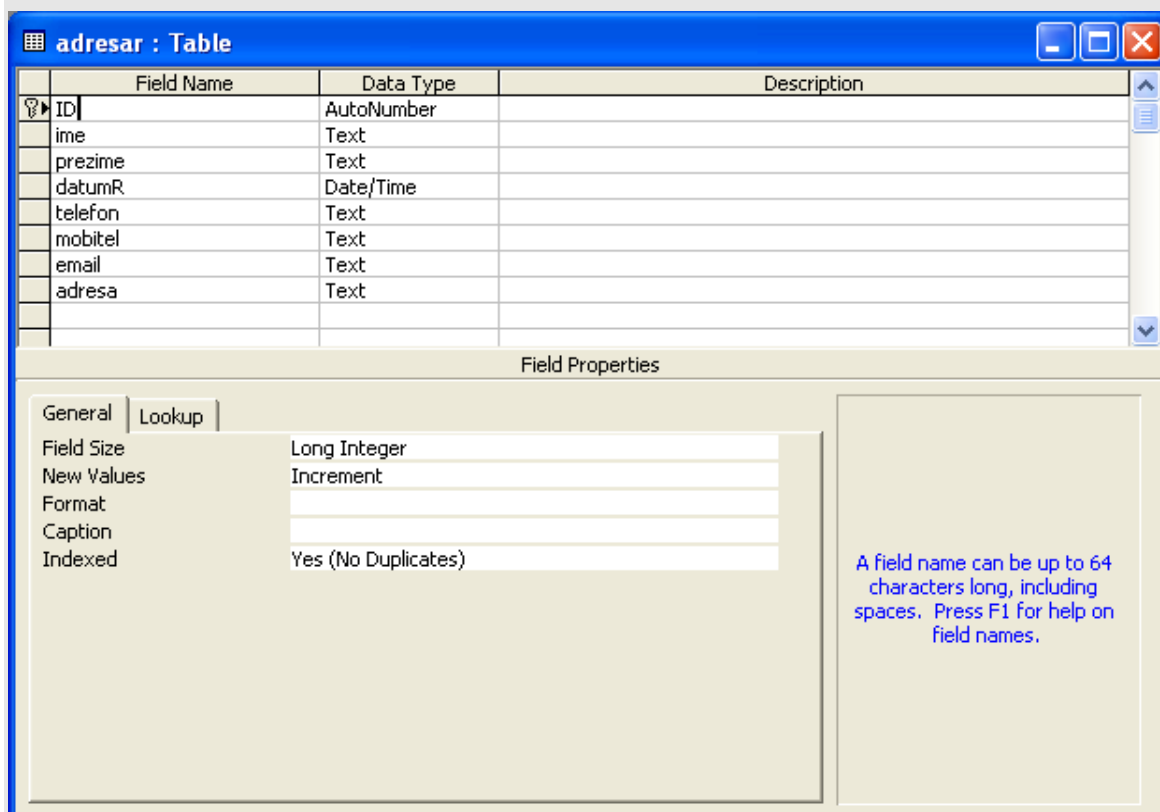
- **prezime** – Text, najviše 30 znakova
- **datumR** – Date/Time, Short Date
- **telefon** – Text, najviše 30 znakova
- **mobitel** – Text, najviše 30 znakova
- **e-mail** – Text, najviše 30 znakova
- **adresa** – Text, najviše 255 znakova

Nadalje ćemo definirati da je **ID** primarni ključ na tablici. To ćemo napraviti tako da kliknemo desnim gumbom miša na stupac **ID** te u izborniku koji se otvori odaberemo **Primary Key**.



Slika 17-6. Definiranje primarnog ključa

Na kraju kliknemo na gumb **Save** odnosno **File → Save**, upišemo ime tablice (*adresar*) te kliknemo na gumb **OK**.



Slika 17-7. Definirana tablica *adresar*

---

## SQL (Structured Query Language)

U tako definiranu tablicu podatke možemo unositi, mijenjati, brisati,... direktno preko sučelja MS Accessa, no nas u stvari zanima nešto drugo. Naime, mi u svojoj aplikaciji nećemo otvarati MS Access i u njega upisivati podatke, mijenjati ih,... već bismo htjeli da to radi naša aplikacija.

Manipulaciju podacima unutar baze podataka omogućuje nam SQL. SQL je skraćenica za *Structured Query Language*, a radi se o jeziku za manipulaciju podacima unutar tablica baze podataka. Četiri su osnovna tipa upita (**Query**):

- **INSERT**
- **SELECT**
- **UPDATE**
- **DELETE**

### Upit **INSERT**

Upit **INSERT** koristit ćemo svaki put kada budemo dodavali podatke u tablicu.

Opći oblik naredbe **INSERT** je:

```
INSERT INTO ime_tablice (st1, st2,... stn)
VALUES (v1, v2,... vn)
```

pri čemu su **st1, st2,... stn**, nazivi stupaca, dok su **v1, v2,... vn** odgovarajuće vrijednosti.

Sve upite ćemo u MS Accessu izvršavati u posebnom prozoru za upite. Za početak otvorimo bazu podataka *adresar.mdb*.

#### Primjer 17 – 2:

Dodajmo u tablicu *adresar* osobu sa sljedećim podacima:

Ime: *Ines*

Prezime: *Marković*

Datum rođenja: *13. 2. 1987.*

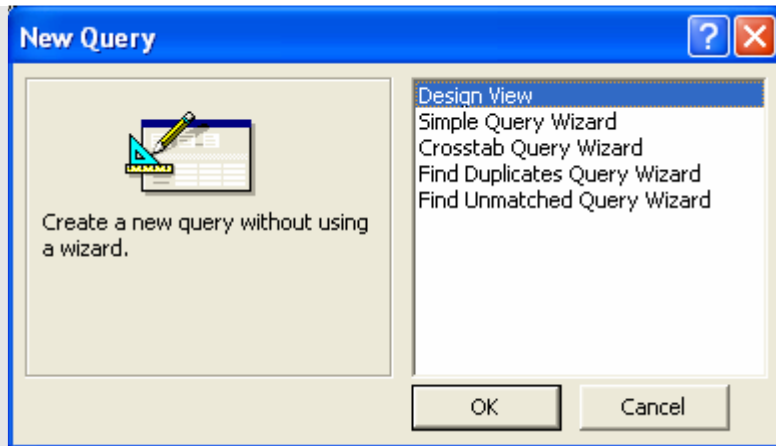
Telefon: *01123456*

Mobitel: *095123456*

Adresa: *V. avenija 23, 10000 Zagreb.*

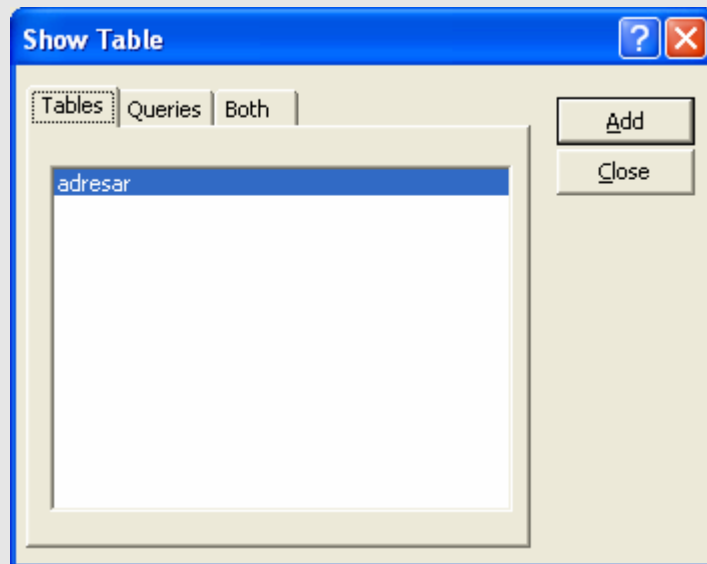
#### Rješenje:

Pokrenimo Access te otvorimo tablicu *adresar.mdb*, koju smo kreirali u prošlom primjeru. U popisu s desne strane odaberemo sličicu **Queries**. U gornjem popisu odaberemo **New** a zatim **Design View**, te kliknemo na **OK**.



Slika 17-8. Odabir načina kreiranja upita

U prozoru koji se pojavi kliknemo na **Close**.



Slika 17-9. Popis tablica za upit

U gornjem dijelu glavnog prozora kliknemo na sličicu **SQL**, te u izborniku koji se otvori odaberemo **SQL View** (**View** → **SQL View**).



Slika 17-10. Odabir sučelja za kreiranje upita

Odabirom ove opcije otvorit će se sljedeći prozor:




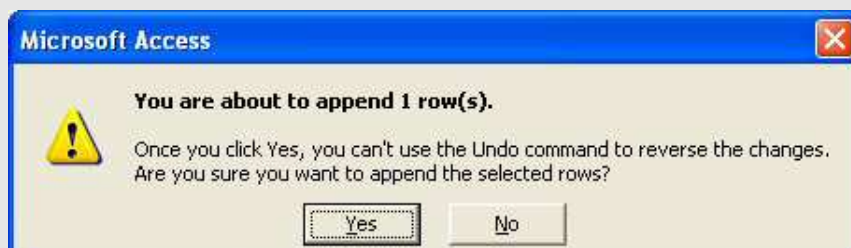


Slika 17-11. Prozor za unos upita

U prozor koji se pojavio upisat ćemo sljedeći upit za dodavanje podataka:

```
INSERT INTO adresar (ime, prezime, datumR, mobitel, telefon, adresa)
VALUES ('Ines', 'Marković', '13.2.1987', '01123456', '095123456',
'V. avenija 23, 10000 Zagreb')
```

Kliknemo na gumb **Execute** (  ) i ukoliko je sve u redu, pojavit će se prozor s porukom:



Slika 17-12. Upit je uspješno izvršen

Otvorimo tablicu **adresar** i u njoj ćemo vidjeti redak s podacima koje smo upravo dodali:

### **Napomena:**

Primijetimo da stupce u upitu nismo naveli onim redoslijedom kojim se nalaze u tablici, bitno je samo da redoslijed stupaca odgovara redoslijedu vrijednosti koje se žele dodati u odgovarajuće stupce tablice.

Isto tako mogli smo primijetiti da jedan stupac u tablici postoji, ali ga nema u upitu, radi se o stupcu *email*. Pogledamo li u tablicu vidjet ćemo da u tom stupcu nema vrijednosti za osobu koju smo unijeli u tablicu, što je i za razumno.

Kao što smo mogli primijetiti u primjeru, vrijednosti koje smo dodavali u tablicu pisali smo unutar jednostrukih navodnika, to je u principu uvijek tako osim kada se dodaju brojevi, koji se ne pišu unutar jednostrukih navodnika. Isto tako je moguće i datume pisati bez navodnika.

## **Upit SELECT**

SELECT ćemo koristiti kada iz tablice budemo htjeli izdvojiti podatke prema nekom kriteriju.

Opći oblik naredbe **SELECT** je:

```
SELECT st1, st2,... stn FROM ime_tablice
WHERE uvjet1 AND/OR uvjet2 AND/OR...AND/OR uvjetm
```

pri čemu su **st1, st2,... stn** nazivi stupaca čije vrijednosti želimo da nam vrati upit, dok je **uvjet1, ..., uvjetm** uvjeti odgovarajući uvjeti koji su najčešće oblika:

- ime\_stupca [operator] vrijednost

pri čemu je operator jedan od relacijskih operatora (=, <, >, <>).

- ime\_stupca like vrijednost

Ovakvim uvjetom najčešće se traže podaci koji "su slični" zadanoj vrijednosti. U tom se slučaju može koristiti zamjenski znak \* koji zamjenjuje jedan ili više znakova.

Moguće je pisati **SELECT** upite i bez uvjeta, u tom slučaju nećemo imati **WHERE**.

Rezultat upita je jedna privremena tablica.

#### Primjer 17 – 3:

Ispišimo imena prezimena, brojeve mobitela te adrese svih osoba koje se zovu **Ines** i prezime započinje slovom **M**.

##### Rješenje:

```
SELECT ime, prezime, mobitel, adresa FROM adresar
WHERE ime = 'Ines' AND prezime LIKE 'M'
```

##### Napomena:

Želimo li da nam upit vrati sve stupce umjesto nabrojavanja svih ćemo jednostavno napisati \*. Npr.

```
SELECT * FROM adresar
```

će nam ispisati sve podatke iz tablice adresar.

## Upit UPDATE

**UPDATE** ćemo koristiti za ažuriranje (izmjenu) podataka u tablici. Opći oblik ove naredbe je:

```
UPDATE ime_tablice SET st1 = vr1, st2 = vr2,... stn = vrn
WHERE uvjet1 AND/OR uvjet2 AND/OR...AND/OR uvjetm
```

Nakon izvršavanja ovog upita, svim zapisima koji zadovoljavaju zadane kriterije bit će, za stupac *st1*, postavljena vrijednost na *vr1*, stupac *st2* će imati vrijednost *vr2*,... dok će stupac *stn* imati vrijednost *vrn*.

#### Primjer 17 – 4:

Izmijenimo podatke u tablici **adresar** tako da svim osobama koji u broju mobitela imaju broj **95** izbrišemo broj mobitela (postavimo ga na prazan string).

##### Rješenje:

```
UPDATE adresar SET mobitel = '' WHERE mobitel LIKE '*95'
```

## Upit DELETE

Kao što je i za pretpostaviti **DELETE** ćemo koristiti kada budemo htjeli brisati čitave zapise iz tablice. Opći oblik naredbe **DELETE** je:

```
UPDATE FROM ime_tablice
WHERE uvjet1 AND/OR uvjet2 AND/OR...AND/OR uvjetm
```

Izvršavanjem ove naredbe iz tablice *ime\_tablice* bit će izbrisani svi podaci koji zadovoljavaju postavljene uvjete.

### Primjer 17 – 5:

Izbrišimo sve osobe iz tablice *adresar*.

Rješenje:

```
DELETE FROM adresar
```

## Java i baze podataka

SQL o kojem smo upravo govorili je standardiziran za rad s različitim tipovima baza podataka (MS Access, MS SQL, ORACLE,...). U principu to znači da ćemo pomoću istog SQL upita dodavati podatke, u bazu kreiranu u Accessu ili MS SQL-u odnosno ORACLU i upravo u tome leži njegova snaga. Da bismo iz vlastitog programa mogli komunicirati s bazom podataka, prvo se moramo spojiti na bazu podataka. E to baš i nije tako jednostavno. Način spajanja ovisi o tipu baze podataka,... što uvelike komplicira stvari.

Međutim, nije baš sve tako crno, u pomoć nam dolazi JDBC. JDBC (Java Database Connectivity) omogućava nam jednostavnu komunikaciju s bazom podataka. Radi se o sučelju za komunikaciju s bazama podataka. Naša aplikacija treba samo poslati odgovarajuće parametre i JDBC će sam obaviti sve potrebne radnje kako bismo uspostavili komunikaciju s bazom podataka i pomoću SQL upita izvršavali određene manipulacije s podacima. Međutim postoji jedan problem, JDBC ne radi sa svim tipovima baza podataka. Osim JDBC-a postoji još i tzv. ODBC (Open Database Connectivity) koje je u stanju spojiti se sa svim tipovima baza podataka. Na žalost, Java ne omogućava spajanje na bazu podataka pomoću ODBC-a, ali zato postoji spona koja je u stanju translirati JDBC u ODBC. Radi se o svojevrsnom mostu između JDBC-a i ODBC-a tzv. **JDBC-ODBC bridge**. Uz puno preznojavanja, muke i priče konačno smo u stanju spojiti se s bilo kojom bazom podataka.

Kako bismo se spojili s bazom podataka pomoću moramo unutar programa napraviti nekoliko predradnji. Prvo što moramo napraviti je reći da želimo komunicirati s ODBC-em preko JDBC-ODBC bridgea. To ćemo napraviti izvršavanjem naredbe:

```
Class.forName ( "sun.jdbc.odbc.JdbcOdbcDriver" ) ;
```

Ukoliko je ova naredba uspješno izvršena spremni smo za kreiranje veze prema bazi podataka, koja će nam omogućiti direktnu komunikaciju s bazom podataka, koja će uključivati izvršavanje SQL upita.

### Povezivanje s bazom podataka

Povezivanje s bazom podataka omogućit će nam metode klase **DriverManager**. **DriverManager** je **static** klasa, a neke od njenih, češće upotrebljavanih metoda su:

**static Connection getConnection (String s)** – na osnovu opisa veze prema bazi podataka (*s*), ova metoda uspostavlja vezu s bazom podataka te vraća objekt tipa **Connection** koji predstavlja konekciju

**static Connection getConnection (String s, String ime, String lozinka)** – na osnovu opisa veze prema bazi podataka (*s*), korisničkog imena (*ime*) i lozinke (*lozinka*), ova metoda uspostavlja vezu s bazom podataka. te vraća objekt tipa **Connection** koji predstavlja konekciju

Ukoliko se spajamo s Access bazom podatak, string *s* koji opisuje vezu prema bazi je sljedećeg oblika:

```
jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=ime.mdb
```

pri čemu je *ime.mdb* čitav put do Accessovog dokumenta u kojem je spremljena naša baza podataka.

Klasa **Connection** sadrži sve potrebne informacije o otvorenoj vezi prema bazi podataka. Neke od metoda ove klase su:

```
void close () – zatvara konekciju prema bazi  
Statement createStatement () – kreira objekt tipa Statement koji će nam omogućiti slanje SQL upita bazi  
boolean isClosed () – vraća true ako je veza prema bazi zatvorena, inače vraća false
```

Za izvršavanje SQL upita na bazi podataka koristit ćemo metode klase **Statement**. Najvažnije metode ove klase su:

```
boolean execute (String s) – izvršava upit (s) tipa INSERT, UPDATE, DELETE  
ResultSet executeQuery (String s) – izvršava upit (s) koji je tipa SELECT i rezultate vraća u obliku objekta tipa ResultSet  
void close () – zatvara Statement objekt
```

Rezultat **SELECT** upita vraća se u obliku **ResultSeta**. **ResultSet** možemo zamisliti kao tablicu u kojoj se nalaze rezultati upita. Nakon što se upit izvrši i rezultati upita se smjeste u **ResultSet**, preostaje nam još pročitati podatke iz **ResultSeta** te ih na neki način prikazati u aplikaciji. Rezultate ćemo čitati tako da s kursorom prolazimo red po red **ResultSeta** i čitamo podatke iz odgovarajućih stupaca unutar trenutnog reda. To će nam omogućiti metode klase **ResultSet**:

```
boolean absolute (int n) – postavlja kursor u n-ti redak ResultSeta  
void afterLast () – postavlja kursor na kraj ResultSeta (iza zadnjeg zapisa)  
void beforeFirst () – postavlja kursor na početak ResultSeta (ispred prvog zapisa)  
void close () – zatvara ResultSeta objekt  
void deleteRow () – briše trenutni redak unutar ResultSeta  
int findColumn (String s) – vraća redni broj stupca s imenom s unutar trenutnog ResultSeta  
boolean first () – postavlja kursor na prvi redak ResultSeta  
boolean getBoolean (String s) – vraća logičku vrijednost koja se nalazi u trenutnom retku u stupcu s imenom s danog ResultSeta  
boolean getBoolean (int n) – vraća logičku vrijednost koja se nalazi u trenutnom retku u stupcu s rednim brojem n danog ResultSeta  
Date getDate (String s) – vraća datum koji se nalazi u trenutnom retku u stupcu s imenom s danog ResultSeta  
Date getDate (int n) – vraća datum koji se nalazi u trenutnom retku u stupcu s rednim brojem n danog ResultSeta  
float getFloat (String s) – vraća realnu vrijednost koja se nalazi u trenutnom retku u stupcu s imenom s danog ResultSeta  
float getFloat (int n) – vraća realnu vrijednost koja se nalazi u trenutnom retku u stupcu s rednim brojem n danog ResultSeta  
int getInt (String s) – vraća cjelobrojnu vrijednost koja se nalazi u trenutnom retku u stupcu s imenom s danog ResultSeta  
int getInt (int n) – vraća cjelobrojnu vrijednost koja se nalazi u trenutnom retku u stupcu s rednim brojem n danog ResultSeta  
int getRow () – vraća redni broj trenutnog retka unutar ResultSeta  
String getString (String s) – vraća tekstualnu vrijednost koja se nalazi u trenutnom retku u stupcu s imenom s danog ResultSeta
```

**String getString (int n)** – vraća tekstualnu vrijednost koja se nalazi u trenutnom retku u stupcu s rednim brojem *n* danog `ResultSeta`  
**boolean isAfterLast ()** – vraća *true* ako je kursor iza zadnjeg reda `ResultSeta`, inače vraća *false*  
**boolean isBeforeFirst ()** – vraća *true* ako je kursor ispred prvog reda `ResultSeta`, inače vraća *false*  
**boolean isFirst ()** – vraća *true* ako je kursor na prvom elementu `ResultSeta`, inače vraća *false*  
**boolean isLast ()** – vraća *true* ako je kursor na zadnjem elementu `ResultSeta`, inače vraća *false*  
**boolean last ()** – postavlja kursor na zadnji redak `ResultSeta`  
**boolean next ()** – pomiče kursor na sljedeći redak `ResultSeta`  
**boolean previous ()** – pomiče kursor na prethodni redak `ResultSeta`  
**boolean relative (int n)** – pomiče kursor za *n* redaka, s obzirom na trenutni redak unutar `ResultSeta`

Sve gore navedene klase za rad s bazama podataka nalaze se unutar posebnog paketa: **java.sql**, koji ćemo morati pozvati unutar aplikacije koja će raditi s bazama podataka.

---

## Još neki elementi grafičkog korisničkog sučelja

Sa aspekta baza podataka znamo sve potrebno za kreiranje aplikacije. Preostaje nam još nešto reći o elementima grafičkog korisničkog sučelja kakve još nismo susreli:

- jahačima,
- liste

Prisjetimo se naša aplikacija treba imati dva sučelja:

- sučelje za unos podataka
- sučelje za pretraživanje, izmjenu i brisanje podataka

Sučelja trebaju biti implementirana pomoću dva jahača na jednom prozoru, a rezultati upita (imena i prezimena osoba) će se ispisivati unutar liste. Klikom na neko ime unutar liste, svi podaci o toj osobi bit će prikazani u posebnim poljima.

### Jahači

Izgled jahača možemo vidjeti na *sluci 17-1*. Kao i za sve ostale elemente Java ima posebnu klasu koja će nam omogućiti kreiranje jahača. Područje na koje ćemo slagati jahače kreirat ćemo klasom **JTabbedPane**. Neki od konstruktora ove klase su:

**JTabbedPane ()** – kreira prazno područje za jahače, pri čemu su jahači inicijalno okrenuti prema gore (`JTabbedPane.TOP`)  
**JTabbedPane (int p)** – kreira prazno područje za jahače, pri čemu je smjer jahača određen parametrom *p*:  
    `JTabbedPane.TOP` – gore  
    `JTabbedPane.BOTTOM` – dolje  
    `JTabbedPane.LEFT` – lijevo  
    `JTabbedPane.RIGHT` – desno

Kao što smo rekli klasa `JTabbedPane` nam omogućava kreiranje područja na koje ćemo stavljati jahače. Jahače, koje ćemo stavljati unutar područja, kreirat ćemo posebnom klasom **`JPanel`**.

Najčešće korišteni konstruktor klase **`JPanel`** je:

**`JPanel ()`** – kreira objekt klase `JPanel`

Između ostalih, klasa `JPanel` nasljeđuje klasu `Container` što znači da je na njega moguće dodavati elemente grafičkog korisničkog sučelja metodom `add ()`, definirati `Layout`,...

`JPanel` ćemo dodati `JTabbedPane` posebnom metodom `addTab ()`. Osim metode `addTab ()` klasa `JTabbedPane` ima i niz drugih metoda, a neke od njih su:

```
void addTab (String s, Component c) – dodaje komponentu c na JTabbedPane pri čemu će naslov komponente biti s  
void addTab (String s, Icon i, Component c) – dodaje komponentu c na JTabbedPane pri čemu će naslov komponente biti s, dok će sličica komponente biti i  
void addTab (String s, Icon i, Component c, String t) – dodaje komponentu c na JTabbedPane pri čemu će naslov komponente biti s, sličica će biti i, dok će tooltip tekst biti t  
int getSelectedIndex () – vraća redni broj trenutno aktivnog jahača  
int getTabCount () – vraća ukupni broj jahača unutar JTabbedPanea  
void remove (int i) – briše jahač s rednim brojem i  
void setColorAt (int i, Color c) – definira pozadinsku boju za jahač s rednim brojem i
```

## Lista

Lista na prvi pogled ima slične karakteristike kao padajuća lista (`JOptionPane`), međutim rad s listama je u Javi znatno složenija od rada s padajućim listama. No to svakako nije razlog da ne kažemo nešto i o njima.

Klasa koja će kreirati listu na ekranu je `JList`. Najčešći konstruktori ove klase su:

**`JList ()`** – kreira objekt klase `JList`

**`JList (ListModel m)`** – kreira objekt klase `JList` koji će prikazivati podatke iz zadanog modela *m*

Najčešće korištene metode klase **`JList`** su:

```
void addListSelectionListener (ListSelectionListener l) – kreira listener na listi koji prati svaku promjenu na označenoj stavki liste  
int getSelectedIndex () – vraća redni broj prve označene stavke unutar liste, ukoliko niti jedna stavka nije označena vraća -1  
int[] getSelectedIndices () – vraća niz sa rednim indeksima svih označenih stavki liste  
Object getSelectedValue () – vraća prvu označenu stavku unutar liste, ukoliko niti jedna stavka nije označena vraća null  
Object[] getSelectedValues () – vraća niz sa vrijednostima svih označenih stavki liste  
boolean isSelectedIndex (int n) – vraća true ako je element na n-tom mjestu označen  
boolean isEmpty () – vraća true ako niti jedan element liste nije označen  
void setModel (ListModel m) – postavlja model m, koji predstavlja sadržaj liste  
void setSelectedIndex (int n) – označava stavku liste na n-tom mjestu
```

```
void setSelectedIndices (int[] a) – označava skup stavki, čiji su redni brojevi elementi niza a  
void setSelectionBackground (Color c) – definira pozadinsku boju označene stavke  
void setSelectionForeground (Color c) – definira boju teksta označene stavke  
void setSelectionInterval (int s, int k) – označava interval stavki
```

Želimo li pratiti događaje na listi, morat ćemo u aplikaciji implementirati i još jedan interface. Radi se o interfeceu **ListSelectionListener**, te ćemo trebati definirati metodu:

```
void valueChanged(ListSelectionEvent e)
```

Kao što možemo primijetiti nema metode kojom ćemo dodati stavku u listu. Ne, nismo ju zaboravili, ona jednostavno ne postoji. Stvar je u tome da je **JList** samo "ljuska" za popis stavki koje će se u njoj nalaziti. Popis stavki koje će biti ispisane u listi nalazi se u posebnom objektu. Mi ćemo stavke držati u objektu koji će biti tipa **DefaultListModel**. **DefaultListModel** je posebna klasa koja će brinuti o sadržaju liste. Samo je jedan konstruktor ove klase:

```
DefaultListModel () – kreira objekt klase DefaultListModel
```

Neke od metoda klase **DefaultListModel** su:

```
void add (int n, Object o) – dodaje objekt na određeno mjesto u listi  
void add (Object o) – dodaje objekt na posljednje mjesto u listi  
void clear () – briše sve stavke liste  
boolean contains (Object o) – provjerava je li o element liste  
boolean contains (Object o) – provjerava je li o element liste  
Object elementAt (int n) – vraća element koji se nalazi na n-tom mjestu liste  
int getSize () – vraća broj elemenata liste  
boolean isEmpty () – provjerava ima li lista elemenata  
void removeElementAt (int n) – iz liste briše element na n-tom mjestu  
Object[] toArray () – vraća niz koji sadrži sve elemente liste u onom redoslijedu u kojem se pojavljuju u listi
```

Sada uistinu imamo sve potrebno za kreiranje aplikacije:



Service	Port	Protocol
echo	7	tcp
echo	7	udp
discard	9	tcp
discard	9	udp
systat	11	tcp
systat	11	tcp
daytime	13	tcp
daytime	13	udp
netstat	15	tcp
qotd	17	tcp
qotd	17	udp
chargen	19	tcp
chargen	19	udp
ftp-data	20	tcp
ftp	21	tcp
telnet	23	tcp
smtp	25	tcp
time	37	tcp
time	37	udp
rlp	39	udp
name	42	tcp
name	42	udp
whois	43	tcp
domain	53	tcp
domain	53	udp
nameserver	53	tcp
nameserver	53	udp
mtp	57	tcp
bootp	67	udp
tftp	69	udp
gopher	70	tcp
rje	77	tcp
finger	79	tcp
http	80	tcp
link	87	tcp
supdup	95	tcp
hostnames	101	tcp
iso-tsap	102	tcp
dictionary	103	tcp
x400	103	tcp
x400-snd	104	tcp

csnet-ns	105	tcp
pop	109	tcp
pop2	109	tcp
pop3	110	tcp
portmap	111	tcp
portmap	111	udp
sunrpc	111	tcp
sunrpc	111	udp
auth	113	tcp
sftp	115	tcp
path	117	tcp
uucp-path	117	tcp
nntp	119	tcp
ntp	123	udp
nbname	137	udp
nbdatagram	138	udp
nbssession	139	tcp
NeWS	144	tcp
sgmp	153	udp
tcprepo	158	tcp
snmp	161	udp
snmp-trap	162	udp
print-srv	170	tcp
vmnet	175	tcp
load	315	udp
vmnet0	400	tcp
sytek	500	udp
biff	512	udp
exec	512	tcp
login	513	tcp
who	513	udp
shell	514	tcp
syslog	514	udp
printer	515	tcp
talk	517	udp
ntalk	518	udp
efs	520	tcp
route	520	udp
timed	525	udp
tempo	526	tcp
courier	530	tcp
conference	531	tcp
rxd-control	531	tcp

netnews	532	tcp
netwall	533	udp
uucp	540	tcp
klogin	543	tcp
kshell	544	tcp
new-rwho	550	udp
remotefs	556	tcp
rmonitor	560	udp
monitor	561	udp
garcon	600	tcp
maitrd	601	tcp
busboy	602	tcp
acctmaster	700	udp
acctslove	701	udp
acct	702	udp
acctlogin	703	udp
acctprinter	704	udp
elcsd	704	udp
acctinfo	705	udp
acctslove2	706	udp
accdisk	707	udp
kerberos	750	tcp
kerberos	750	udp
kerberos_master	751	tcp
kerberos_master	751	udp
passwd_server	752	udp
userreg_server	753	udp
krb_prop	754	tcp
erlogin	888	tcp
kpop	1109	tcp
phone	1167	udp
ingreslock	1524	tcp
maze	1666	udp
nfs	2049	udp
knetd	2053	tcp
eklogin	2105	tcp
rmt	5555	tcp
mtb	5556	tcp
man	9535	tcp
w	9536	tcp
mantst	9537	tcp
bnews	10000	tcp
rscs0	10000	udp

queue	10001	tcp
rscs1	10001	udp
poker	10002	tcp
rscs2	10002	udp
gateway	10003	tcp
rscs3	10003	udp
remp	10004	tcp
rscs4	10004	udp
rscs5	10005	udp
rscs6	10006	udp
rscs7	10007	udp
rscs8	10008	udp
rscs9	10009	udp
rscsa	10010	udp
rscsb	10011	udp
qmaster	10012	tcp
qmaster	10012	udp
irc	6667	tcp