U program „chat“ napravljeno je grafičko sučelje prema korisniku te je uz kauzalni uređaj dodan mehanizam međusobnog isključivanja koji spriječava da dva korisnika mogu poslati poruku u isto vrijeme

Chat program za konferencije

Goran Brajdić

12.7.2015.

UVOD

U ovom projektu cilj je implementacija konferencijskog chat programa u Javi. Postupno ćemo ga dograđivati grafički, kako bi bio pristupačniji korisniku, te funkcionalno, mehanizmom međusobnog isključivanja te kauzalnog uređaja.

Objasnimo prvo svojstvo kauzalnog uređaja među porukama za koje želimo da bude osigurano u našoj *chat* – aplikaciji, te sami algoritam za osiguravanje kauzalnog urađaja.

Traži se da bilo koje dvije poruke, koje možda putuju po različitim kanalima ali prema istom procesu, budu primljene u onom redosljedu kako su bile poslane. Neka → predstavlja uređaj „dogodilo se prije“, koji uspoređuje događaje iz istih ili različitih procesa. Neka su s_1 i s_2 događaji slanja poruka, a r_1 i r_2 odgovarajući događaji primanja poruka.

Stoga definicija svojstva kauzalnog uređaja tada glasi:

- ako $s_1 \rightarrow s_2$ tada nije istina da $r_2 < r_1$ (CO)

gdje < predstavlja uređaj događaja unutar istog procesa.

Opišimo sada algoritam koji preuređuje poruke kako bi osigurao njihov kauzalni reosljed. Pretpostavljamo da proces nikad ne šalje poruku samom sebi. Svaki proces ima cjelobrojnu matricu $M[\,][\quad]$. Element $M[j][k]$ u procesu P_i bilježi broj poruka koje je proces P_j poslao procesu P_k prema saznnjima od P_i . Kad god proces P_i pošalje poruku procesu P_j, element $M[i][j]$ u procesu P_i se povećava za 1. Matrica $M[\,][\quad]$ se šalje kao prilog uz poruku. Kad god proces P_i primi novu poruku, najprije se provjerava je li ta poruka kvalificirna za isporuku. Ako poruka još nije kvalificirana, ona se sprema u buffer do trenutka kad će postati kvalificirana.

Poruka m s priloženom matricom $W[\,][\,]$ koju je proces P_i primio od procesa P_j kvalificirana je ako vrijedi $W[\,j][\,i] = M[j][i] + 1$. Naime u suprotnom slučaju postojala bi poruka koju je proces P_j poslao procesu P_i prije m i koja je prije od m na redu za isporuku. Za svaki $k \neq j$ vrjedi da je $M[k][i] \geq W[k][i]$. Naime, kad bi bilo $W[k][i] > M[k][i]$, to bi značilo da postoji poruka koja je poslana prije m, a još nije bila isporučena. Kad god je poruka konačno prihvaćena za isporuku, informacija u $M[\,][\quad]$ se ažurira s podacima iz $W[\,][\quad]$, dakle uzmaju se maksimumi po odgovarajućim elementima. Struktura kauzalne poruke zadana je klasom *CausalMessage.* Algoritam za kauzalno uređivanje implementiran je pomoću klase *CausalLinker.*

```
public class CausalMessage {
    Msg m;
    int N;
    int W[][];
    public CausalMessage(Msg m, int N, int matrix[][]) {
```

```
        this.m = m;
        this.N = N;
        W = matrix;
    }
    public int[][] getMatrix() {
        return W;
    }
    public Msg getMessage() {
        return m;
    }
}
```

```
import java.util.*; import java.net.*; import java.io.*;
public class CausalLinker extends Linker {
    int M[][];
    LinkedList deliveryQ = new LinkedList(); // deliverable messages
    LinkedList pendingQ = new LinkedList(); // messages with matrix
    public CausalLinker(String basename, int id, int numProc)
                                            throws Exception {
        super(basename, id, numProc);
        M = new int[N][N]; Matrix.setZero(M);
    }
    public synchronized void sendMsg(int destId, String tag, String msg){
        M[myId][destId]++;
        super.sendMsg(destId, "matrix", Matrix.write(M));
        super.sendMsg(destId, tag, msg);
    }
    public synchronized void multicast(IntLinkedList destIds,
                                            String tag, String msg) {
        for (int i=0; i<destIds.size(); i++)
            M[myId][destIds.getEntry(i)]++;
        for (int i=0; i<destIds.size(); i++) {
            int destId = destIds.getEntry(i);
            super.sendMsg(destId, "matrix", Matrix.write(M));
            super.sendMsg(destId, tag, msg);
        }
    }
    boolean okayToRecv(int W[][], int srcId) {
        if (W[srcId][myId] > M[srcId][myId]+1) return false;
        for (int k = 0; k < N; k++)
            if ((k!=srcId) && (W[k][myId] > M[k][myId])) return false;
        return true;
    }
    synchronized void checkPendingQ() {
        ListIterator iter = pendingQ.listIterator(0);
        while (iter.hasNext()) {
            CausalMessage cm = (CausalMessage) iter.next();
            if (okayToRecv(cm.getMatrix(), cm.getMessage().getSrcId())){
                iter.remove(); deliveryQ.add(cm);
            }
        }
    }


    // polls the channel given by fromId to add to the pendingQ
    public Msg receiveMsg(int fromId) throws IOException {
        checkPendingQ();
        while (deliveryQ.isEmpty()) {
            Msg matrix = super.receiveMsg(fromId);// matrix
```

```
            int [][]W = new int[N][N];
            Matrix.read(matrix.getMessage(), W);
            Msg m1 = super.receiveMsg(fromId);//app message
            pendingQ.add(new CausalMessage(m1, N, W));
            checkPendingQ();
        }
        CausalMessage cm = (CausalMessage) deliveryQ.removeFirst();
        Matrix.setMax(M, cm.getMatrix());
        return cm.getMessage();
    }
}
```

Klasa `CausalLinker` proširuje klasu `Linker` kako bi mogla raditi s porukama koje su proširene matricama. Metoda `sendMsg()` povećava $M[id][destId]$ da bi uzela u obzir dotičnu poruku, te prilaže ažuriranu $M[\,][\,]$ uz poruku.

Metoda `multicast()` služi za slanje iste poruke većem broju primatelja. Najprije se poaćava $M[id][destId]$ za sve $destId$ za zadane liste primatelja. Zatim se tako ažurirana $M[\,][\,]$ šalje kao prilog uz svaku kopiju poruke.

Metoda `okayToRecv()` određuje je li poruka kvalificirana za isporuku aplikaciji.

Metoda `receiveMsg()` koristi dvije vezane liste (reda) za spremanje poruka:

- Lista `deliveryQ` sadrži sve poruke koje su kvalificirane i mogu se isporučiti.
- Lista `pendingQ` sprema sve poruke koje su primljene, no nisu još kvalifiirane za isporuku.

Metoda `checkPendingQ()` prolazi listom `pendingQ` da bi provjerila je li neka poruka u njoj postala kvalificirana. Pronađena kvalificirana poruka se iz `pendingQ` prebacuje u `deliveryQ`.

Kad aplikacijski sloj pozivom `receiveMsg()` zatraži poruku:

- Prvo se pozove `checkPendingQ()` da bi se eventualne kvalificirane poruke prebacile u `deliveryQ`.
- Ako je `deliveryQ` iapk prazna, poziva se blokirajuća metoda `super.receiveMsg()` da bi se dočekao dolazak sasvim nove poruke.
- Kad dođe ta noa poruka, ona se najprije smješta u `pendingQ`, pa se ponovo poziva `checkPendingQ()` da bi se ta poruka eventualno prebacila u `deliveryQ`.
- Kad `deliveryQ` nije prazan, skida se i isporučuje prva poruka iz te liste, a matrica $M[\,][\quad]$ se ažurira da bi se zabilježila ta isporuka.

Sama *chat* – aplikacija u kojoj korisnik može poslati poruke većem broju korisnika realizirana je kao klasa `Chat`.

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

public class Chat extends Process {
    public Chat(Linker initComm) {
        super(initComm);
```

```
    }
    public synchronized void handleMsg(Msg m, int src, String tag){
        if (tag.equals("chat")) {
            System.out.println("Message from " + src +":");
            System.out.println(m.getMessage());
        }
    }
    public String getUserInput(BufferedReader din) throws Exception {
        System.out.println("Type your message in a single line:");
        String chatMsg = din.readLine();
        return chatMsg;
    }
    public IntLinkedList getDest(BufferedReader din) throws Exception {
        System.out.println("Type in destination pids with -1 at end:");
        System.out.println("Only one pid for synch order:");
        IntLinkedList destIds = new IntLinkedList(); //dest for msg
        StringTokenizer st = new StringTokenizer(din.readLine());
        while (st.hasMoreTokens()) {
            int pid = Integer.parseInt(st.nextToken());
            if (pid == -1) break;
            else destIds.add(pid);
        }
        return destIds;
    }
    public static void main(String[] args) throws Exception {
        String baseName = args[0];
        int myId = Integer.parseInt(args[1]);
        int numProc = Integer.parseInt(args[2]);
        Linker comm = null;
        if (args[3].equals("simple"))
            comm = new Linker(baseName, myId, numProc);
        else if (args[3].equals("causal"))
            comm = new CausalLinker(baseName, myId, numProc);
        else if (args[3].equals("synch"))
            comm = new SynchLinker(baseName, myId, numProc);
        Chat c = new Chat(comm);
        for (int i = 0; i < numProc; i++)
            if (i != myId) (new ListenerThread(i, c)).start();
        BufferedReader din = new BufferedReader(
        new InputStreamReader(System.in));
        while (true) {
            String chatMsg = c.getUserInput(din);
            if (chatMsg.equals("quit")) break;
            IntLinkedList destIds =  c.getDest(din);
            if (args[3].equals("synch"))
                comm.sendMsg(destIds.getEntry(0), "chat", chatMsg);
            else
                comm.multicast(destIds, "chat", chatMsg);
        }
    }
}
```

Program u petlji učitava od korisnika tekst poruke i listu identifikatora primatelja poruke. Zatim se poruka šalje svim primateljima. Program završava onda kada korisnik upiše poruku *„quit“*.
Pri pokretanju programa kao dodatni argument komandnog retka zadaje se uređaj za poruke koji želimo osigurati. To može biti:

- FIFO uređaj implementiran standardnom klasom `Linker`.
- Kauzalni uređaj implementiran klasom `CausalLinker`.
- Sinkroni uređaj implementiran klasom `SynchlLinker`

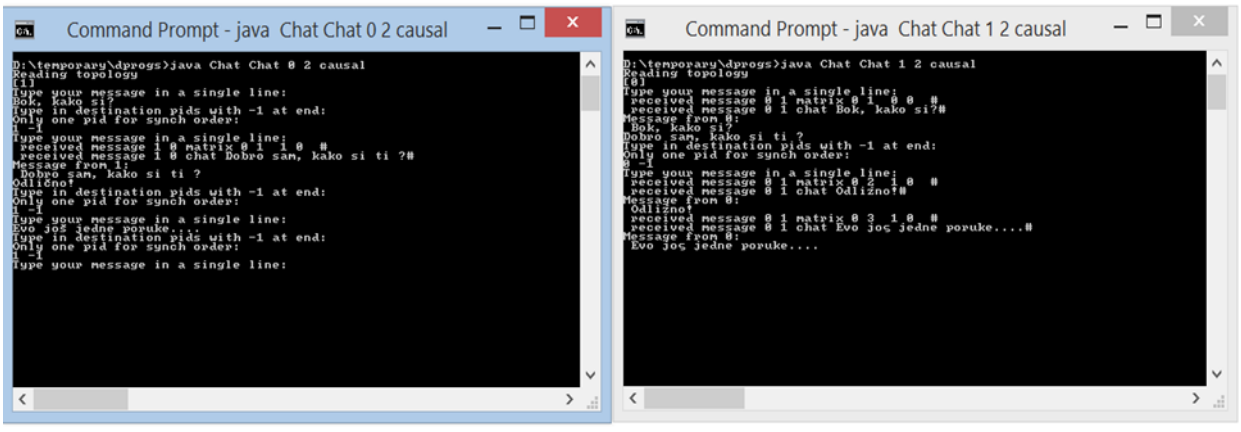
Metoda `handleMsg()` služi za ispis primljene poruke od drugih sudionika na ekran.

Metoda `getUserInput()` čita liniju poruke koju sudionik želi poslati, sprema je u string `chatMsg`, te vraća taj string.

Metoda `getDest()` zadužena je za unos destinacije za poruku koju sudionik želi poslati. Unose se `pid`-ovi procesa (tj identifikatori procesa kojima želimo poslti poruku). F-ja vraća vezanu listu `Int`-ova u kojoj se nalaze `pid`-ovi.

Ako *chat* – aplikacija koristi FIFO uređaj, dakle klasu `Linker`, tad je mogu sljedeći scenarij. Proces P_0 istovremeno šalje isti upit proesima P_1 i P_2. Proces P_1 šalje odgovor na upit procesima P_0 i P_2. P_2 je primio odgovor prije nego što je primio sam upit. Ako aplikacija koristi kauzalni uređaj, dakle klasu `CausalLinker`, tada gornji scenarij nije moguć. Ako aplikacija koristi sinkroni uređaj, dakle klasu `SynchlLinker`, tada gornji scenarij pogotovo nije moguć. Algoritam za kinkroni uređaj ne može obavljati multicast.

Cilj ovog projekta je ugraditi mehanizam međusonog isključivanja koji pretpostavlja kauzalni uređaj među porukama. Mehanizmom međusobnog isključivanja ćemo spriječiti situaciju da dva sudionika pišu poruku u isto vrijeme, dok ćemo kauzalnim uređajem spriječiti već navedenu situaciju, tj da proces P_2 najprije vidi odgovor procesa P_1 na pitanje procesa P_0, prije nego što je vidio samo pitanje procesa P_0. Sljedeće poboljšanje je dobiveno ugradnjom boljeg korisničkog slučelja (GUI-a). Naime, osnovna chat aplikacija bazirana je na sučelju komandne linije (CLI, Slika1), gdje se narebe zadaju tekstualnim linijama. To je rezultiralo nepreglednim čitanjem, unosom i prikazom poruka sudionika *chat* – aplikacije.



Slika 1: Command line interface (CLI)

OSTVARENJE MEĐUSOBNOG ISKLJUČIVANJA NAD KAUZALNIM UREĐAJEM

U ovom poglavlju detaljnije ćemo objasniti implementaciju mahanizma međusobnog isključivanja sa kauzalnim uređajem nad porukama. Cilj nam je spriječiti situaciju da dva sudionika šalju poruku u isto vrijeme. Sve izmjene za ostvarenje mađusobnog isključivanja nad izvornom *chat* – aplikacijom napravljene su u klasi `CausalLinker`.

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

public class CausalLinker extends Linker {

    DirectClock v;
    static int[] q; // request queue
    int M[][];
    LinkedList deliveryQ = new LinkedList(); // deliverable messages
    LinkedList pendingQ = new LinkedList(); // messages with matrix

    public CausalLinker(String basename, int id, int numProc)
                                              throws Exception {
        super(basename, id, numProc);
        M = new int[N][N]; Matrix.setZero(M);
        v = new DirectClock(N, myId);
        q = new int[N];
        for (int j = 0; j < N; j++)
            q[j] = 0;
    }

    public synchronized void multicast(IntLinkedList destIds,
                                              String tag, String msg) {
        for (int i=0; i<destIds.size(); i++){
            M[myId][destIds.getEntry(i)]++;
        }

        for (int i=0; i<destIds.size(); i++) {
        v.tick();
        q[myId] = v.getValue(myId);
        while(!okayCS())
            myWait();

            int destId = destIds.getEntry(i);
            super.sendMsg(destId, "matrix", Matrix.write(M));
            super.sendMsg(destId, tag, msg);
       }
    }

    boolean okayToRecv(int W[][], int srcId) {
        if (W[srcId][myId] > M[srcId][myId]+1)
        return false;
        for (int k = 0; k < N; k++)
            if ((k!=srcId) && (W[k][myId] > M[k][myId]))
            return false;
            v.receiveAction(srcId, v.getValue(srcId));
        return true;
    }


synchronized void checkPendingQ() {
        ListIterator iter = pendingQ.listIterator(0);
        while (iter.hasNext()) {
            CausalMessage cm = (CausalMessage) iter.next();
            if (okayToRecv(cm.getMatrix(), cm.getMessage().getSrcId())){
                iter.remove(); deliveryQ.add(cm);
```

```
                if(!(okayCS())){
                q[cm.getMessage().getSrcId()]++;
                notify();

                }
            }
        }
    }
    // polls the channel given by fromId to add to the pendingQ
    public Msg receiveMsg(int fromId) throws IOException {

        checkPendingQ();

        while (deliveryQ.isEmpty()) {
            Msg matrix = super.receiveMsg(fromId);// matrix
            int [][]W = new int[N][N];
            Matrix.read(matrix.getMessage(), W);
            Msg m1 = super.receiveMsg(fromId);//app message
            pendingQ.add(new CausalMessage(m1, N, W));
            checkPendingQ();
        }
        CausalMessage cm = (CausalMessage) deliveryQ.removeFirst();
        Matrix.setMax(M, cm.getMatrix());
        return cm.getMessage();
    }

    boolean okayCS() {
        for (int j = 0; j < N; j++){
            if (isGreater(q[myId], myId, q[j], j))
                return false;
            if (isGreater(q[myId], myId, v.getValue(j), j))
                return false;
        }
        return true;
    }

    boolean isGreater(int entry1, int pid1, int entry2, int pid2) {

        return (((entry1 == entry2) && (pid1 > pid2)));
    }

    public synchronized void myWait() {
        try {
            wait();
        }
        catch (InterruptedException e) { System.err.println(e); }
    }


}
```

Da bi međusobno isključivanje bilo moguće, prva modifikacija koja je napravljena je uvođenje sata neposredne ovisnosti `v` koji je implementiran u klasi `DirectClock`, te reda `q` za spremanje zahtjeva za resursom, koje procesi moraju održavati.

Sljedeća modifikacija napravljena je uvođenjem nekih metoda iz drugih klasa koje su nam bile potrebe.

Metoda `myWait()`, uvedena iz klase `Process` stavlja dretvu na čekanje, dretva će „spavati" sve dok je ne probudi poziv `notify()` pozvana iz metode `checkPendingQ()`. Metode `myWait()` i `checkPendingQ()` su sinkronizirane.

Pomoćna metoda `isGreater()` vratit će `true` ako su vrijednosti vremenskih žigova jednaki i ako je `pid1 > pid2`. Inaće vraća `false`. To nam je jedino i potrebo provjeravati, kako je zadovoljen kauzalni uređaj, moramo spriječiti samo situaciju kada procesi šalju poruku u isto vrijeme, tj. kada imaju iste vremenske žigove.

Metoda `okayCS()` vraća bool vrijednost, tj koristi pomoćnu metodu `isGreater()` za usporedbu vremenskih žigova i identifikatora procesa.

Te tri uvedene metode služiti će za modifikaciju dvaju već postojećih metoda `multicast()` i `checkPendingQ()`.

Metoda `multicast()` koja služi za slanje iste poruke većem broju primatelja modificirana je dodavanjem sljedećih funkcionalnosti:

- Proces koji šalje poruku poveća vrijednost svog sata `v`, te sprema tu vrjednost u svoj red zahtjeva `q`.
- Sve dok uvjet `okayCS()` ne postane `true`, poziva se metoda `myWait()`, koja poziva čekanje dretve. Ukoliko dva procesa šalju poruku u isto vrijeme, proces s većim `pid`-om morati će čekati da proces sa manjim `pid`-om pošalje poruku.

Metoda `checkPendingQ()` koja provjereva da li je neka poruka u listi `pendingQ` postala kvalificirana, modificirana je na sljdedeći način:

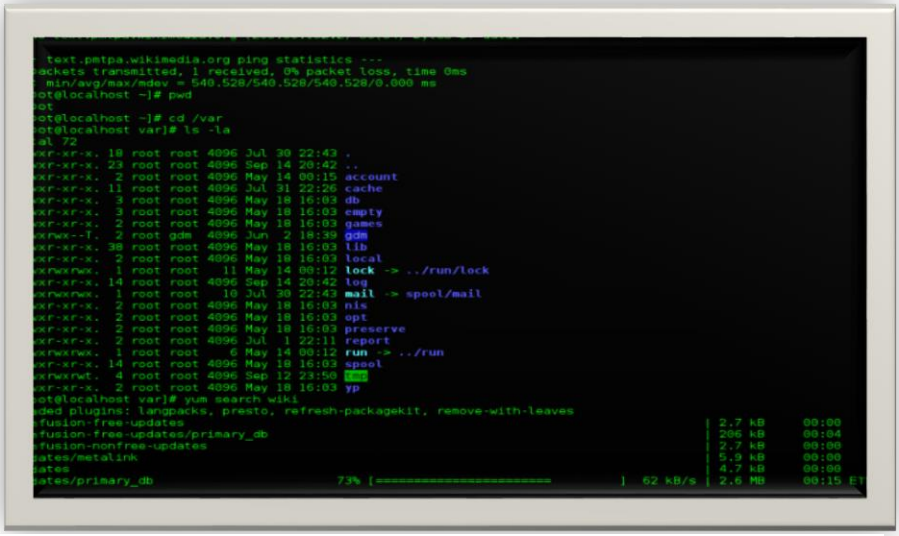
- Ako uvjet `okayCS()` nije `true`, povećamo za jedan žig zahtjeva (u redu zahtjeva) procesa koji je poslao poruku (tj. koji je izašao iz kritične sekcije).
- Zatim pozivamo metodu `notify()` koja će prekinuti čekanje dretve koja je pozvala `myWait()`.

Time je gore navedenim modifikacijama ostvaren mehanizam međusobnog isključivanja, nad kauzalnim uređajem. Sada se ne može dogoditi da dva sudionika šalju poruku u isto vrijeme, jer će jedan od sudionika (onaj s većim identifikatorom) morati čekati da prvo poruku pošalje drugi sudionik (onaj s manjim identifikatorom). A zbog mehanizma preuređivanja redosljeda poruka kojim se ostvaruje kauzalni uređaj biti će spriječena situacija u kojoj sudionik C najprije vidi odgovor sudionika B na pitanje sudionika A prije nego što je vidio samo pitanje sudionika A.

GRAFIČKO KORISNIČKO SUČELJE

U računarstvu, grafičko korisničko sučelje ili kratko GUI(graphical user interface) je tip sučelja koji omogućuje korisnicima da imaju interakciju s elektroničkim uređajima koristeći grafičke ikone i vizualne pokazatelje kao sekundarnu notaciju, nasuprot tekstualno baziranim sučeljima, gdje je sve prikazano tekstom te se tekst također koristi za davanje naredbi programu. Ono je nastalo kao reakcija na sporo učenje korisnika kako koristiti CLI(command-line interface) tj sučelje komande linije u kojem se naredbe unose tekstualnim linijama.
Novi Windows-i su primjer programa koji ima grafičko korisničko sučelje a MS-DOS ili naši dosadašnji programi u Javi su primjer programa koji imaju sučelje komandne linije.

Vizualna komponenta je važan dio razvoja softwerskih aplikacija u području interakcije čovjeka i računala. Cilj GUI-ja je poboljšanje efikasnosti i olakšavanje korištenja programa koji se skriva iza njega. Niz elemenata koji zajedno čine vizualni jezik koristi se za prikazivanje informacija pohranjenih u računalu. Najčešća kombinacija takvih elemenata je WIMP(window, icon, menu, pointing device) tj prozor, ikona, meni i pokazivač. WIMP stil interakcije koristi neki ulazni uređaj za kontrolu pokazivača, najčešće miš, i prikazuje informacije organizirane u prozore koji su prikazani ikonama. Naredbe se nalaze u menijima i pokreću pokazivačem.



CLI

WIMP

GUI U JAVI

Java ima dva API-ja za kreiranje GUI-ja, AWT(Abstract Windowing Toolkit) i Swing. Oni su dio JFC-a(Java Foundation Classes) grafičkog framework-a za kreiranje korisničkog sučelja koji ne ovisi o operativnom sustavu na kojem se pokreće.

AWT

AWT je uveden u JDK 1.0. i čini jezgru kreiranja GUI-ja u Javi. Koristi se za stvaranje WIMP-a, događaja (events) i layout manager-a (raspored elemenata koji ne ovisi o veličini prozora ili rezoluciji ekrana) te sadrži osnove grafičke elemente kao što su naprimjer button, text box i frame. Sastoji se od 12 paketa od kojih su najkorišteniji java.awt i java.awt.event.
java.awt sadrži osnovne klase:

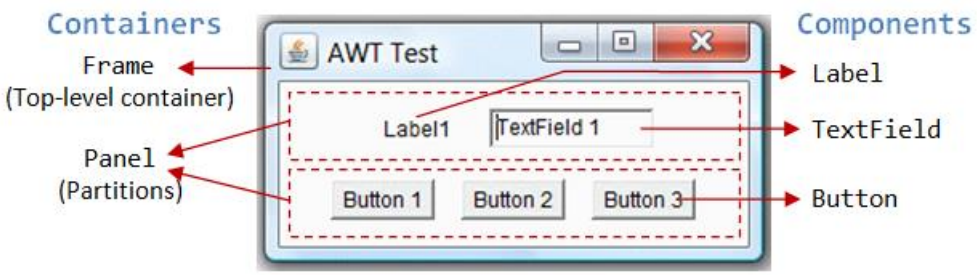
- GUI Component classes (kao što su Button, TextField i Label)
- GUI Container classes (kao što su Frame, Panel, Dialog i ScrollPane)
- Layout managers (kao što su FlowLayout, BorderLayout i GridLayout)
- Custom graphics classes (kao što su Graphics, Color i Font)

Java.awt.event sadrži klase za upravljanje događajima:

- Event classes (kao što su ActionEvent, MouseEvent, KeyEvent i WindowEvent)
- Event Listener Interfaces (kao što su ActionListener, MouseListener, KeyListener i WindowListener)
- Event Listener Adapter classes (kao što su MouseAdapter, KeyAdapter, i WindowAdapter)

java.awt

Komponente se moraju nalaziti unutar spremnika (container-a) i raspoređene su na određeni način (specific layout-grid, flow…). Komponenti i spremnika može biti više a no mora postojati „glavni“ spremnik, najčešće korišteni su Frame, Dialog i Applet. Komponente se dodaju pomoću funkcije *add (Component c)* koja se poziva na spremniku. Ako imamo spremnik *aContainer* i želimo dodati komponentu *aComponent* u njega napisali bi *aContainer.add (aComponent).*



Raspored komponenti odnosno layout poziva se na spremnicima na način *aContainer.setLayout(aLayout)* gdje je aLayout neki željeni raspored.

java.awt.event

Java koristi „Event-Driven“ (ili „Event-Delegation“) model za upravljanje događajima. U takvom modelu korisnik koristeći neku ulaznu jedinicu (npr miš ili tipkovnicu) pokreće kod koji će ovisno o tome što je korisnik napravio izvršiti određenu naredbu. Model sadrži tri stvari: izvor(kao što su Button ili Textfield), slušać/r (listener/s) i događaj. Korisnik interakcijom s izvorom pomoću neke ulazne jedinice stvara događaj. Taj događaj se onda prosljeđuje svim slušačima koji se odnose na njega i zatim se izvršava odgovarajuća naredba. Slušaći se dodaju izvoru slično kao što se komponente dodaju spremniku, *aSource.addXxxListener(alistener),* gdje Xxx treba zamijeniti s vrstom slušaća npr „Mouse“. Jedan primjer kako se slušać kreira dan je ispod:

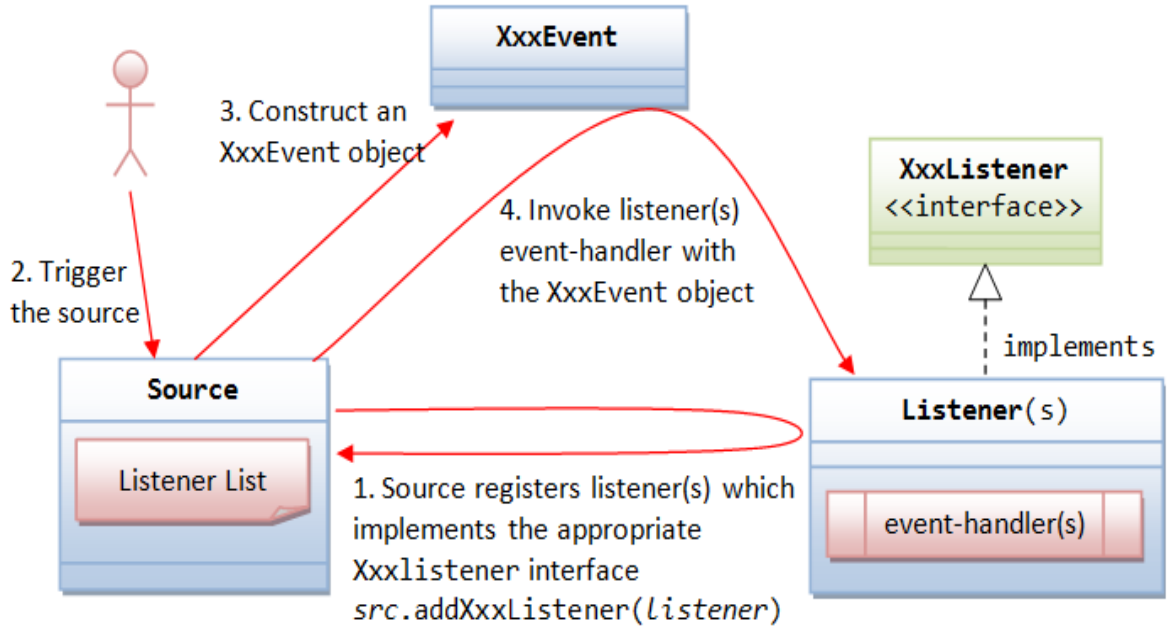
```
public interface MouseListener {
   public void mousePressed(MouseEvent evt);  // pozvana dok „držimo“ miš
   public void mouseReleased(MouseEvent evt); // pozvana nakon što je miš otpušten
   public void mouseClicked(MouseEvent evt);  //pozvana nakon što je miš kliknut
   public void mouseEntered(MouseEvent evt);  // pozvana kad se pokazivač miša našao
na komponenti
   public void mouseExited(MouseEvent evt);    // pozvana kad se pokazivač miša
maknuo sa komponente}

class MyMouseListener implement MouseListener {
   @Override
   public void mousePressed(MouseEvent e)  {
      System.out.println("Mouse-button pressed!");
   }

   @Override
   public void mouseReleased(MouseEvent e) {
      System.out.println("Mouse-button released!");
   }

   @Override
   public void mouseClicked(MouseEvent e)  {
      System.out.println("Mouse-button clicked (pressed and released)!");
   }

   @Override
   public void mouseEntered(MouseEvent e)  {
      System.out.println("Mouse-pointer entered the source component!");
   }

   @Override
   public void mouseExited(MouseEvent e)   {
      System.out.println("Mouse exited-pointer the source component!");
   }
}
```

Sada bi za Button aButton da mu dodamo gornji slušać pisali
aButton.addMouseListener(MyMouseListener);

Znači kad kreiramo slušaća moramo preopteretiti sve njegove metode u slučaju kada nam to netreba tj. recimo da bi htjeli samo preopteretiti metodu za klik miša onda možemo koristiti „Listener Adapter Classes“ umjesto „Listener Interfaces“. One nam dopuštaju da preopteretimo samo funkcije koje želimo a ostale funkcije su defaultno *null*(apstraktna klasa). Tada bi umjesto gornjeg koda za klik miša na aButton pisali sljedeće:

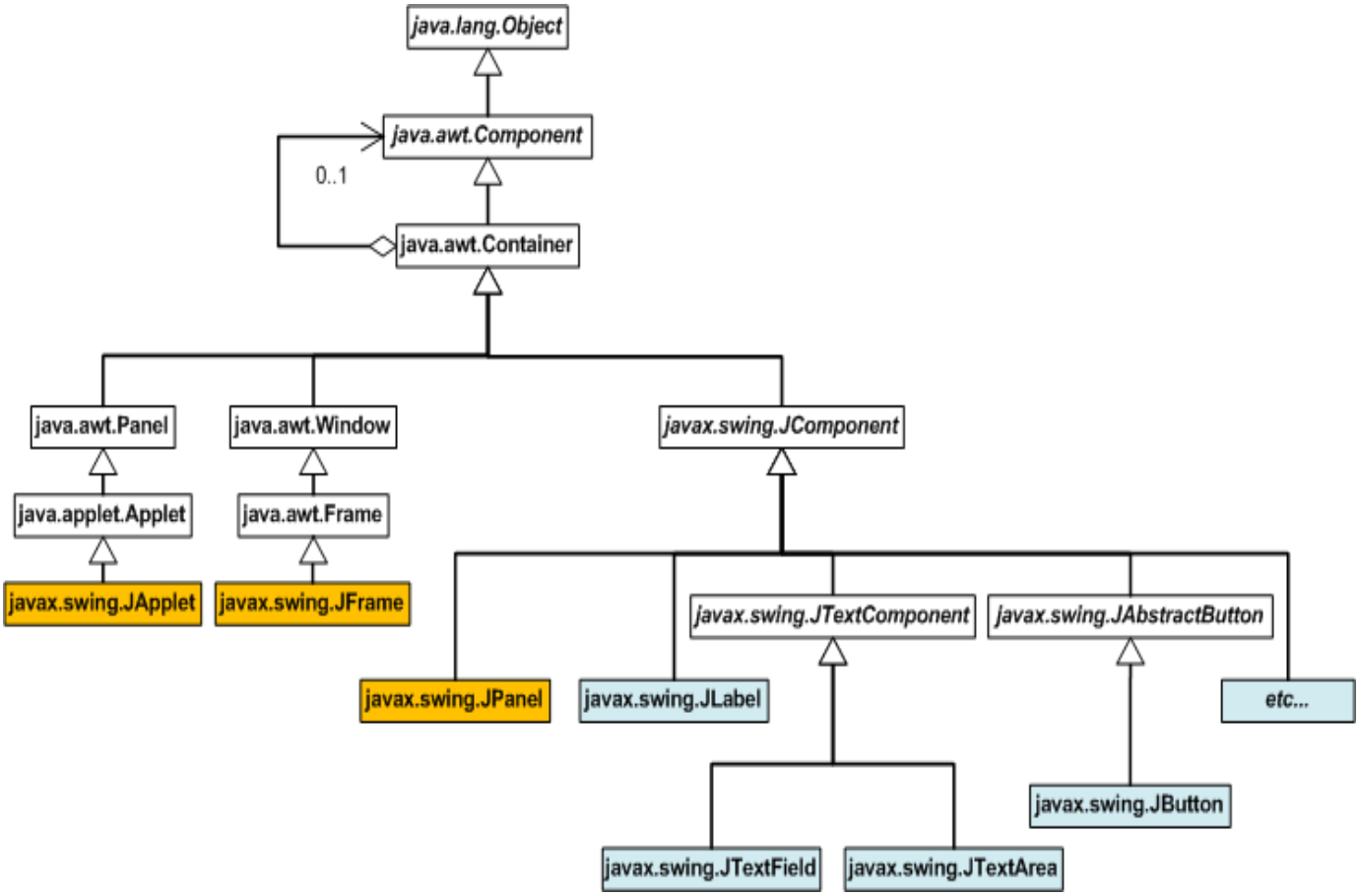
```
aButton.addMouseListener(new MouseAdapter(){
   @Override
   public void mouseClicked(MouseEvent e)  {
      System.out.println("Mouse-button clicked (pressed and released)!");
   }
```

Za kraj prikazujemo sliku kako Java upravlja događajima.



SWING

Swing je uveden u JDK 1.1. i sadrži sofisticiranije komponente od njegovog prethodnika AWT-a kao što su npr. ScrollPane, Tree, List i Table. Sadrži 18 paketa od kojih je nakorišteniji javax.swing. Napisan je u čistoj javi stoga je 100% neovisan o platformi na kojoj se izvodi. Njegove komponente podržavaju „pluggable-look-and-feel“ što znači da možemo mijenjati izgled tijekom izvršavanja programa. Npr. „Swing button“ koji se izvršava na Windows-u izgledat će kao tipičan „Windows button“, a ako se izvršava na UNIX-u izlgedat će kao „UNIX button“, tj možemo prilagoditi izgled operacijskom sustavu na kojem pokrećemo GUI. Swing koristi neke stvari iz AWT-a, klase za upravljanje događajima (java.awt.event), te „layout manager“ klase iz java.awt no on ima i neke nove koji se nalaze u javax.swing. Imena njegovih komponenti počinju sa prefiksom „J“ (JButton, JTextField…) a način pisanja koda isti je ko kod AWT-a. Ispod se nalazi jedan dijagram odnosa klasa često korištenih komponenti.



GUI U CHAT-u

Kao što smo vidjeli naš chat ima CLI koji je nepregledan i kompliciran za upotrebu. Kreirat ćemo grafičko sučelje koje će biti intuitivnije i lijepše prikazivati poruke svih korisnika. GUI je implementiran u klasi MyFrame, čiji kod prikazujemo ispod.

```
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Color;
import java.awt.Font;
import java.awt.Point;
import javax.swing.border.Border;
import javax.swing.BorderFactory;
import javax.swing.border.BevelBorder;
import javax.swing.plaf.basic.BasicScrollBarUI;

//klasa koja opisuje GUI
//sve će se nalaziti u glavnom spremniku MyFrame
public class MyFrame extends JFrame{
    //stvara JTextArea veličine 10×47 koji služi za prikaz primljenih i
poslanih poruka
    JTextArea ChatBox=new JTextArea(10,47);
    //kreira novi JScrollPane koji se sastoji od ChatBox-a i
    //vertikalnog i horizontalnoh ScrollBar-a za ChatBox koji se uvijek
prikazuje
    JScrollPane myChatHistory=new
JScrollPane(ChatBox,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
    //stvara JTextArea veličine 5×40 koji služi za unošenje poruka koje
korisnik želi poslati
    JTextArea UserText = new JTextArea(5,40);
    //kreirano novi JScrollPane koji se sastoji od UserText-a i
    //vertikalnog i horizontalnog ScrollBar-a za UserText koji se prikazuje
    //ako je uneseni tekst veći od granica UserText-a
    JScrollPane myUserHistory=new
JScrollPane(UserText,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
    //stvara novi gumb Send s imenom "Send to"
    JButton Send = new JButton("Send to");
    //stvara novi JTextField koji služi za unošenje imena korisnika kojima
se želi poslati poruka
    JTextField User=new JTextField(38);
    //stvara novi gumb Quit s imenom "Quit"
    JButton Quit = new JButton("Quit");
    //varijabla koja signalizira je li korisnik poslao poruku ili nije
    //tj je li kliknuo Send ili ne, u početku inicijalizirana kao false.
    boolean OKPressed = false;
    //varijabla u koju će se spremati poruka koja se treba poslati
    //odnosno tekst unešen u UserText
    String poruka;
```

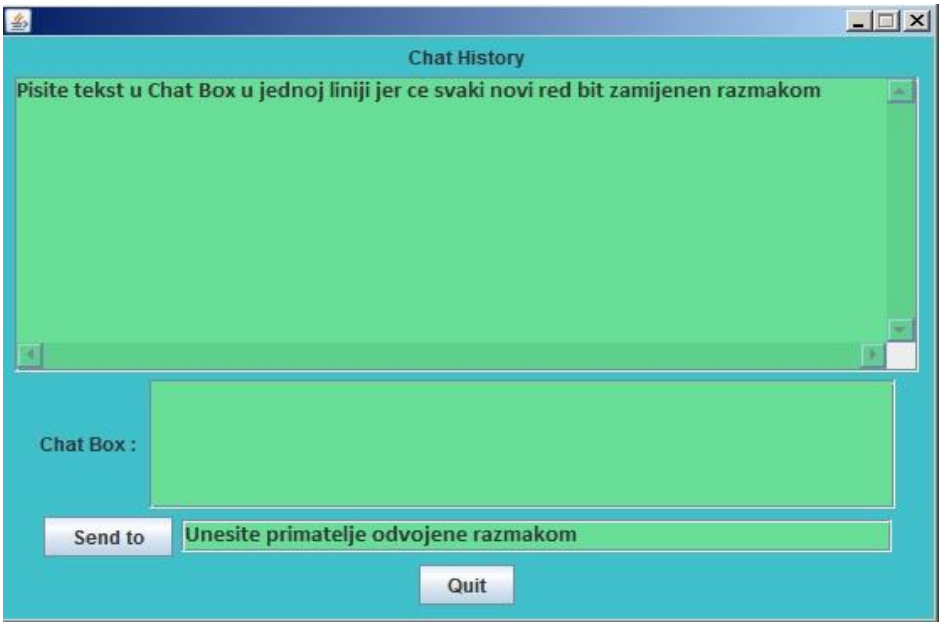
```
    //varijabla u koju će se spremati imena korisnika kojima se želi
poslati poruka
    //odnosno tekst unešen u User
    String userpid;
    //varijabla koja će služiti za pohranjivanje koordinata pokazivača miša
    static Point mouseDownCompCoords=null;

    //konstrukor za GUI odnosno glavni prozor
    public MyFrame(){
        //funkcija koja spriječava mijenjanje veličine prozora
        setResizable(false);
        //funkcija koja postavlja veličinu prozora na 560×400
        setSize(560,400);

        setUndecorated(true);
        getRootPane().setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createBevelBorder(BevelBorder.RAISED),
            BorderFactory.createBevelBorder(BevelBorder.LOWERED)));

        //dohvaćamo "unutrašnjost" prozora, u njega možemo dodavati
komponente
        Container cp=getContentPane();
        //kreiramo boje koje ćemo kasnije dodijeliti određenim komponentama
        Color c=new Color(64,192,203);
        Color d=new Color(104,222,151);
        Color e=new Color(95,209,141);
        Color g=new Color(79,172,116);
        Color f=new Color(204,204,204);
        //postavljamo boju cp-a na "c"
        cp.setBackground(c);
        //postavljamo layout na "FlowLayout.CENTER"
        //"FlowLayout" postavlja komponente u redoslijedu kojem su dodane
        //s lijeva na desno dok više nema mjesta, tada pređe u "novi red"
        //"CENTER" znači da će biti pozicionirane u sredini
        cp.setLayout(new FlowLayout(FlowLayout.CENTER));
        //kreira i dodaje JLabel "Chat History" u cp
        cp.add(new JLabel("Chat History"));
        //postavlja boju ChatBox-a na "d"
        ChatBox.setBackground(d);
        //kreira novi font "Calibri" podeblja ga i postavi ga na veličinu
14
        Font font=new Font("Calibri", Font.BOLD, 14);
        //postavljamo font ChatBox-a na "font"
        ChatBox.setFont(font);
        //dodaje tekst u ChatBox
        ChatBox.append("Pisite tekst u Chat Box u jednoj liniji jer ce
svaki novi red bit zamijenen razmakom \n");
        //Onemogućavamo korisniku da editira ili dodaje tekst u ChatBox
        ChatBox.setEditable(false);
        //postavlja pozadinsku boju ScrollBar-ova ChatBox-a na "e"
        myChatHistory.getVerticalScrollBar().setBackground(e);
        myChatHistory.getHorizontalScrollBar().setBackground(e);
        //postavlja boju gumba koji nastaje kada tekst izađe iz
        //granica ChatBox-a na "g"
        myChatHistory.getVerticalScrollBar().setUI(new BasicScrollBarUI(){
            @Override
            protected void configureScrollBarColors(){
                this.thumbColor = g;
        }});
        myChatHistory.getHorizontalScrollBar().setUI(new
BasicScrollBarUI(){
```

```
            @Override
            protected void configureScrollBarColors(){
                this.thumbColor = g;
        }});

myChatHistory.getHorizontalScrollBar().getComponent(0).setBackground(e);

myChatHistory.getHorizontalScrollBar().getComponent(1).setBackground(e);

myChatHistory.getVerticalScrollBar().getComponent(0).setBackground(e);

myChatHistory.getVerticalScrollBar().getComponent(1).setBackground(e);

        JTextField kut=new JTextField(1);
        kut.setBackground(e);
        kut.setBorder(BorderFactory.createEmptyBorder());
        kut.setEditable(false);
        myChatHistory.setCorner(JScrollPane.LOWER_RIGHT_CORNER,kut);


        //dodaje myChatHistory u cp
        cp.add(myChatHistory);
        //kreira i dodaje novi JLabel "Chat Box: " u cp
        cp.add(new JLabel("Chat Box : "));
        //na sličan način radimo sada dizajn "UserText"-a i njegovih
"ScrollBar"-ova
        UserText.setBackground(d);

        myUserHistory.getVerticalScrollBar().setBackground(e);
        myUserHistory.getHorizontalScrollBar().setBackground(e);

        myUserHistory.getVerticalScrollBar().setUI(new BasicScrollBarUI(){
            @Override
            protected void configureScrollBarColors(){
                this.thumbColor = g;
        }});
        myUserHistory.getHorizontalScrollBar().setUI(new
BasicScrollBarUI(){
            @Override
            protected void configureScrollBarColors(){
                this.thumbColor = g;
        }});


myUserHistory.getHorizontalScrollBar().getComponent(0).setBackground(e);

myUserHistory.getHorizontalScrollBar().getComponent(1).setBackground(e);

myUserHistory.getVerticalScrollBar().getComponent(0).setBackground(e);

myUserHistory.getVerticalScrollBar().getComponent(1).setBackground(e);
        //dodaje "myUserHistory" u cp
        cp.add(myUserHistory);
        //dodaje gumb "Send" u cp
        cp.add(Send);
        //postavljamo boju, font i tekst u "User"
        User.setBackground(d);
        User.setFont(font);
        User.setText("Unesite primatelje odvojene razmakom");
        //dodaje "User" u cp
        cp.add(User);
```

```
        //dodaje gumb "Quit" u cp
        cp.add(Quit);

        Send.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                poruka=(String)UserText.getText();
                poruka=poruka.replace("\n", " ");
                userpid=User.getText();
                OKPressed = true;
                UserText.setText(null);
            }
        });

        //Kreiramo događaj koji na klik gumba Quit prekine izvršavanje
programa
        Quit.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                System.exit(0);
            }
        });
        //Kreiramo događaj koji na klick JTextField-a "User" izbriše
postojeći tekst
        User.addMouseListener(new MouseAdapter(){
            @Override
            public void mouseClicked(MouseEvent e){
                User.setText(null);
            }
        });

        //sljedeći kod omogućava da pomičemo prozor po ekranu kad god
držimo i "vučemo" miš
        addMouseListener(new MouseAdapter(){
            @Override
            public void mouseReleased(MouseEvent e) {
                 mouseDownCompCoords = null;
            }
            @Override
            public void mousePressed(MouseEvent e) {
                mouseDownCompCoords = e.getPoint();
            }
        });
        addMouseMotionListener(new MouseAdapter(){
            @Override
            public void mouseDragged(MouseEvent e) {
                Point currCoords = e.getLocationOnScreen();
                setLocation(currCoords.x - mouseDownCompCoords.x,
currCoords.y - mouseDownCompCoords.y);
            }
        });

        //naredba koja ako zatvorimo prozor, prekine izvršavanje programa
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //da bi se "MyFrame" prikazao na ekranu treba ga učiniti vidljivim
        //to čini sljedeća naredba
        setVisible(true);
    }
}
```

Većina koda objašnjena je u komentarima, sada ćemo još objasniti ono što nije.

Svaki JFrame ima defaultni izlged kao „Javin prozor“ tj izgleda ovako:



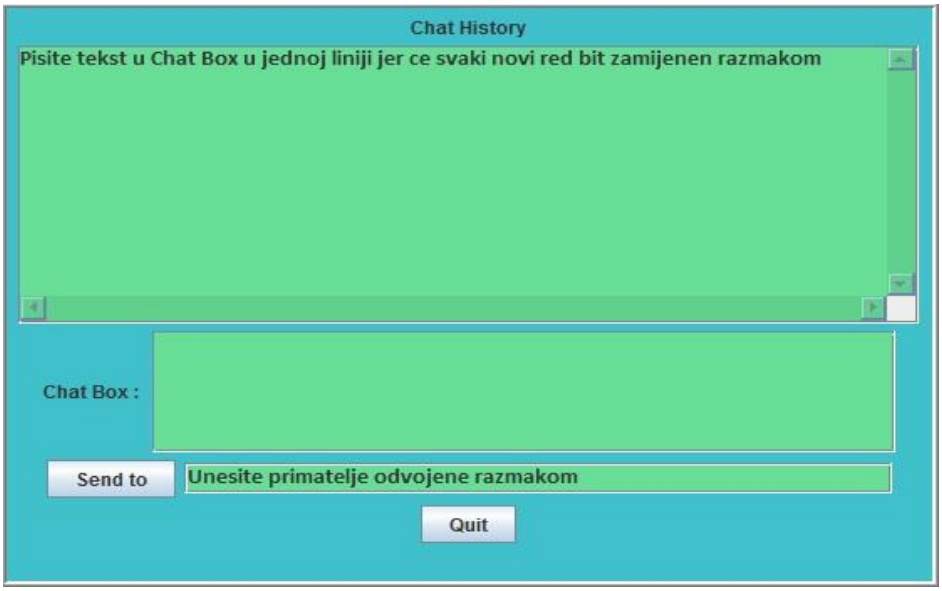
Ima plavi rub sa slikom kave i sivim gubima tj jako izgledom podsjeća na Windows aplikaciju, da bi uklonili taj rub pišemo u konstruktoru od MyFrame

```
setUndecorated(true);
```

Te stvaramo valastiti rub sa sljedećemo naredbom

```
getRootPane().setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createBevelBorder(BevelBorder.RAISED),
            BorderFactory.createBevelBorder(BevelBorder.LOWERED)));
```

Sada naš chat izgleda ovako:



Kod kreiranja „ScrollBar“-ova ako su horizontalni i vertikalni jedan do drugog stvara se rupa koja se u javi zove Corner. Mada je vidljiv on nije komponenta i na njemu nemožemo ništa raditi osim dodavati komponente, stoga da bi mu pridružili boju, kreirat ćemo novu komponentu obojati je u određenu boju i staviti je u tu prazninu. To radi sljedeći kod:

```
JTextField kut=new JTextField(1);
        kut.setBackground(e);
        kut.setBorder(BorderFactory.createEmptyBorder());
        kut.setEditable(false);
        myChatHistory.setCorner(JScrollPane.LOWER_RIGHT_CORNER,kut);
```

Granice komponente smo izbrisali, tj kreirali „prazne“ granice i zabranili editiranje tog područja kako nebi dobili dojam da je to zaseban dio.

Ostaje još za objasniti što se događa kada se klikne gumb „Send“, tj što radi sljedeći kod:

```
Send.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                poruka=(String)UserText.getText();
                poruka=poruka.replace("\n", " ");
                userpid=User.getText();
                OKPressed = true;
                UserText.setText(null);
            }
        });
```

Tekst koji je korisnik unio u „UserText“ odnosno u područje označeno sa „ChatBox“ sprema se u varijablu „poruka“. S obzirom da je način slanja poruke implementiran tako da se dohvaća jedna linija, svaki novi red zamjenjujemo razmakom inače bi došlo do greške u slučaju kad bi korisnik unio novi red. U varijablu „userpid“ spremaju se primatelji, odnosno primatelji koji su uneseni u „User“ tj prazno polje desno od gumba „Send to“. Varijabla „OKPressed“ se postavlja na true, jel ona signalizira je li gumb kliknut ili nije. Te se nakon toga „poslana“ poruka briše kako bi korisnik mogao unijeti novu bez potrebe da ju „ručno“ briše.

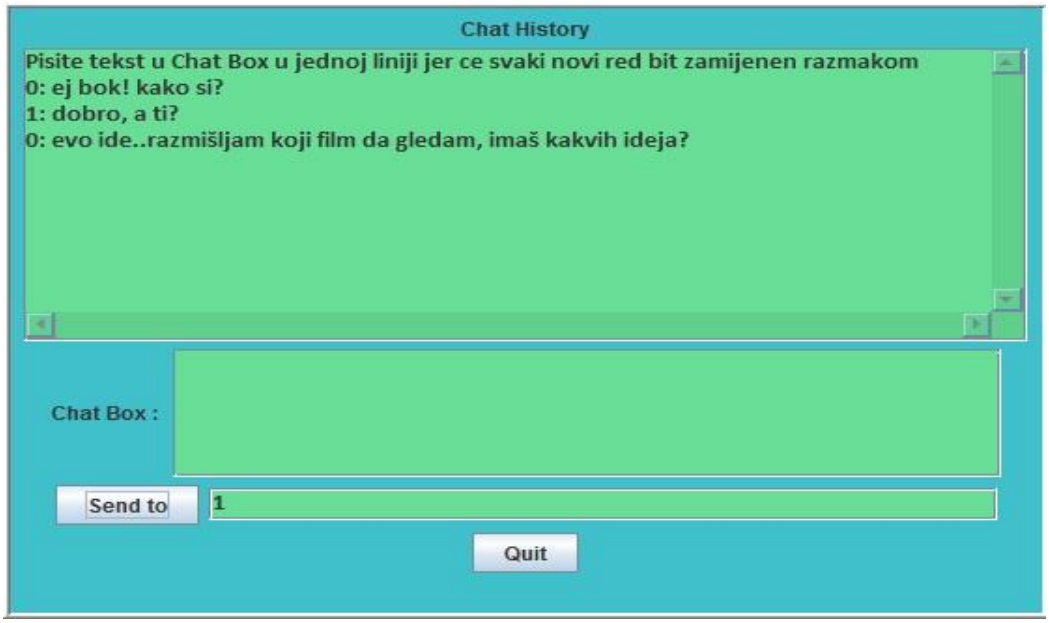
S ovime smo završili objašnjavanje implementacije GUI-ja, no sada trebamo uspostaviti komunikaciju između našeg Chat-a i njega. Kao što smo već djelomično objasnili, korisnik poruke koje želi poslati upisuje u područje nazvano „Chat Box“ a korisnike kojima želi poslati poruku u područje desno od gumba „Send to“. Nakon klika gumba ako je poruka uspješno poslana pojaviti će se u području nazvanim „Chat History“. Da bi navedno ostvarili moramo promijeniti implementaciju klase Chat. Program se pokreće u „command prompt“-u naredbom „java Chat x y“ gdje x treba zamijeniti id-om procesa tj „vlastitim imenom“ a y sa ukupnim brojem procesa. Nakon toga otvoriti će se GUI.
Same promjene kojima smo to ostvarili upisat ćemo u komentarima koda koji prikazujemo ovdje:

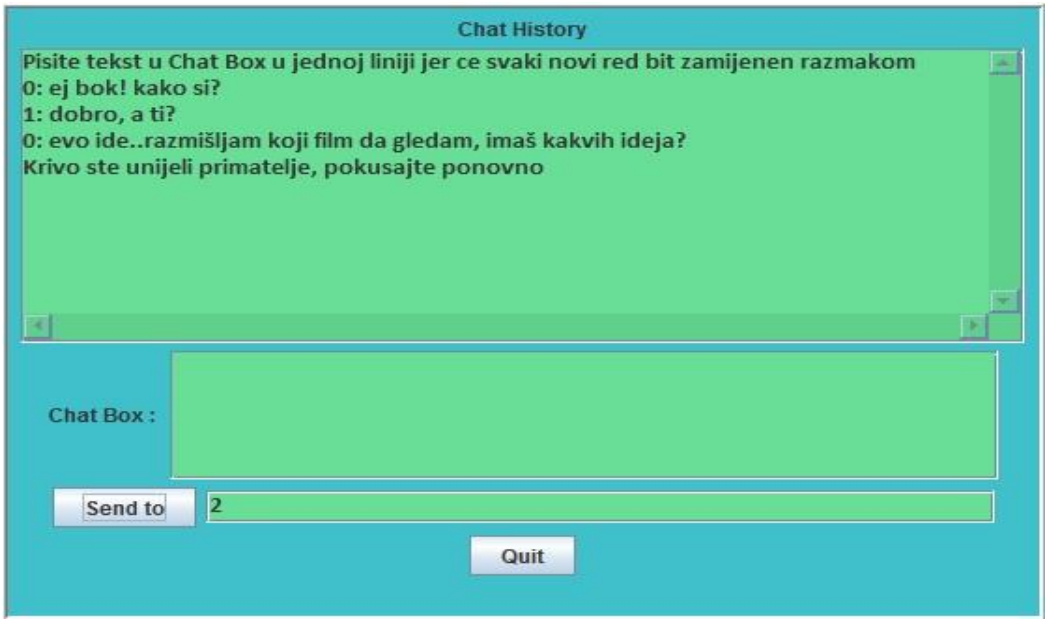
```
import java.io.*; import java.util.*;

public class Chat extends Process {

    //stvaramo GUI
    static MyFrame t=new MyFrame();

    public Chat(Linker initComm) {
        super(initComm);
    }
    public synchronized void handleMsg(Msg m, int src, String tag){
```

```
        if (tag.equals("chat")) {
            //ispiši korisnika i poruku koju je poslao na ekran tj područje
            //u GUI-ju označeno sa Chat History primatelja
            t.ChatBox.append(src + ":" + m.getMessage() + "\n");
        }
    }
    public String getUserInput() throws Exception {
        //dok nije kliknut gumb "Send" spavaj, tj ne radi ništa
        while (t.OKPressed != true){Thread.sleep(500);}
        //kada je kliknut gumb dohvati vrijednost varijable poruka i spremi
je u chatMsg
        //tj dohvati poslanu poruku
        String chatMsg=t.poruka;
        return chatMsg;
    }
    public IntLinkedList getDest(BufferedReader din) throws Exception {
        IntLinkedList destIds = new IntLinkedList(); //dest for msg
        //dok nije kliknut gumb "Send" spavaj, tj ne radi ništa
        while (t.OKPressed != true){Thread.sleep(500);}
        //kada je kliknut gumb dohvati vrijednost varijable userdpid, dodaj
joj -1
        //i spremi je u st, tj dohvati listu primatelja poruke
        StringTokenizer st = new StringTokenizer(t.userpid +" -1");
        while (st.hasMoreTokens()) {
            int pid=Integer.parseInt(st.nextToken());
            if (pid == -1) break;
            else destIds.add(pid);
        }
        return destIds;
    }
    public static void main(String[] args) throws Exception {
        String baseName = "Chat";
        int myId = Integer.parseInt(args[0]);
        int numProc = Integer.parseInt(args[1]);
        Linker comm = null;
        comm = new CausalLinker(baseName, myId, numProc);
        Chat c = new Chat(comm);
        for (int i = 0; i < numProc; i++)
            if (i != myId) (new ListenerThread(i, c)).start();
        BufferedReader din = new BufferedReader(
        new InputStreamReader(System.in));
        while (true){
            System.out.println(c.getUserInput());
            String chatMsg = c.getUserInput();
            //probaj dohvatiti listu primatelja, postoji mogućnost da
            //korisnik ne unese primatelje ili unese nepostojeće
            try{
                IntLinkedList destIds=c.getDest(din);
                comm.multicast(destIds, "chat", chatMsg);
                //ispiši svoje ime i poslanu poruku na vlastiti ekran tj
                //područje u GUI-ju označeno sa Chat History
                t.ChatBox.append(myId + ": " + chatMsg +"\n");
                //signaliziraj da gumb Send više nije kliknut
                //tj da je obrađena poruka
                t.OKPressed=false;
            }
            //ako je "try" neuspješan ispiši poruku o grešci te
signaliziraj
            //da gumb Send više nije kliknut tj da je obrađena
poruka(neuspješno
            //doduše)
```

```
            catch(Exception e){
                t.ChatBox.append("Krivo ste unijeli primatelje, pokusajte
ponovno \n");
                t.OKPressed=false;
            }
        }
    }
}
```

Za kraj još prikazujemo par slika slika konačnog izgleda i funkcionalnosti naše Chat aplikacije. Slike prikazuju razgovor između korisnika 0 i 1, prvotno dobro poslanih poruka, a zatim što se ispiše ako korisnik 1 pokuša poslati poruku nepostojećem korisniku 2.





Literatura

Vijay K. Garg, Concurrent and Distributed Computing in Java

http://web.studenti.math.pmf.unizg.hr/~manger/protect/DP-07.pdf

http://web.studenti.math.pmf.unizg.hr/~manger/protect/DP-04.pdf

http://web.studenti.math.pmf.unizg.hr/~manger/protect/DP-02.pdf

https://en.wikipedia.org/wiki/Command-line_interface

https://en.wikipedia.org/wiki/WIMP_(computing)

https://en.wikipedia.org/wiki/Graphical_user_interface

https://en.wikipedia.org/wiki/Java_Foundation_Classes

https://en.wikipedia.org/wiki/Abstract_Window_Toolkit

https://en.wikipedia.org/wiki/Swing_(Java)

https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

https://www.clear.rice.edu/comp310/JavaResources/GUI/