

## Apêndice A

### Receita de Código Final

**Cyber BeatBox**

Bass Drum	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Start
Closed Hi-Hat	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Stop
Open Hi-Hat	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tempo Up
Acoustic Snare	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Tempo Down
Crash Cymbal	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	serializelt
Hand Clap	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	restore
High Tom	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	dance beat
Hi Bongo	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Andy: groove #2
Maracas	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Chris: groove2 revised
Whistle	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nigel: dance beat
Low Conga	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Cowbell	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Vibraslap	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Low-mid Tom	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
High Agogo	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Open Hi Conga	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Sua mensagem será enviada para os outros participantes, junto com seu padrão de batida atual, quando você pressionar "sendIt".

Mensagens recebidas dos participantes. Clique em uma para carregar o padrão que veio com ela e, em seguida, clique em 'Start' para reproduzi-lo.

Finalmente, a versão completa da BeatBox!

Ela se conectará com um objeto MusicServer simples para que você possa enviar e receber padrões de batida de outros clientes.

## O programa final do cliente da Beat Box

Grande parte desse código é o mesmo das Receitas de Código dos capítulos anteriores, portanto, não comentaremos tudo novamente. As novas partes incluem:

**GUI** — dois novos componentes foram adicionados à área de texto que exibe mensagens recebidas (na verdade uma lista de rolagem) e ao campo de texto.

**REDE** — da mesma forma que o SimpleChatClient deste capítulo, a BeatBox agora se conecta com o servidor e captura um fluxo de entrada e saída.

**SEGMENTOS** — novamente, da mesma forma que o SimpleChatClient, iniciamos uma classe de 'leitura' que procura continuamente mensagens recebidas no servidor. Mas em vez de apenas texto, as mensagens recebidas incluem DOIS objetos: a mensagem em forma de String e a ArrayList serializada (o objeto que contém o estado de todas as caixas de seleção).

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

public class BeatBoxFinal {

    JFrame theFrame;
    JPanel mainPanel;
    JList incomingList;
    JTextField userMessage;
    ArrayList<JCheckBox> checkboxList;
    int nextNum;
    Vector<String> listVector = new Vector<String>();
    String userName;
    ObjectOutputStream out;
    ObjectInputStream in;
    HashMap<String, boolean[]> otherSeqsMap = new HashMap<String, boolean[]>();

    Sequencer sequencer;
    Sequence sequence;
    Sequence mySequence = null;
    Track track;

    String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat", "Open Hi-Hat", "Acoustic Snare", "Crash Cymbal", "Hand Clap", "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga", "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo", "Open Hi Conga"};

    int[] instruments = {35, 42, 46, 38, 49, 39, 50, 60, 70, 72, 64, 56, 58, 47, 67, 63};

    public static void main (String[] args) {
        new BeatBoxFinal().startUp(args[0]); // args[0] is your user ID/screen name
    }

    public void startUp(String name) {
        userName = name;
        // abre uma conexão com o servidor
        try {
            Socket sock = new Socket("127.0.0.1", 4242);
            out = new ObjectOutputStream(sock.getOutputStream());
            in = new ObjectInputStream(sock.getInputStream());
            Thread remote = new Thread(new RemoteReader());
            remote.start();
        } catch (Exception ex) {
            System.out.println("couldn't connect - you'll have to play alone.");
        }
        setUpMidi();
        buildGUI();
    } // fecha startUp

    public void buildGUI() {
```

Adiciona um argumento de linha de comando para o nome de sua tela.  
Exemplo: % java BeatBoxFinal theFlash

Nada de novo... Configura a rede, a E/S e cria (e inicia) o segmento de leitura.

Código de GUI, nada de novo aqui.

```

theFrame = new JFrame("Cyber BeatBox");
BorderLayout layout = new BorderLayout();
JPanel background = new JPanel(layout);
background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

checkboxList = new ArrayList<JCheckBox>();

Box buttonBox = new Box(BoxLayout.Y_AXIS);
JButton start = new JButton("Start");
start.addActionListener(new MyStartListener());
buttonBox.add(start);

JButton stop = new JButton("Stop");
stop.addActionListener(new MyStopListener());
buttonBox.add(stop);

JButton upTempo = new JButton("Tempo Up");
upTempo.addActionListener(new MyUpTempoListener());
buttonBox.add(upTempo);

JButton downTempo = new JButton("Tempo Down");
downTempo.addActionListener(new MyDownTempoListener());
buttonBox.add(downTempo);

JButton sendIt = new JButton("sendIt");
sendIt.addActionListener(new MySendListener());
buttonBox.add(sendIt);

userMessage = new JTextField();
buttonBox.add(userMessage);

incomingList = new JList();
incomingList.addListSelectionListener(new MyListSelectionListener());
incomingList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane theList = new JScrollPane(incomingList);
buttonBox.add(theList);
incomingList.setListData(listVector); // nenhum dado com o qual iniciar

Box nameBox = new Box(BoxLayout.Y_AXIS);
for (int i = 0; i < 16; i++) {
    nameBox.add(new Label(instrumentNames[i]));
}

background.add(BorderLayout.EAST, buttonBox);
background.add(BorderLayout.WEST, nameBox);

theFrame.getContentPane().add(background);
GridLayout grid = new GridLayout(16,16);
grid.setVgap(1);
grid.setHgap(2);
mainPanel = new JPanel(grid);
background.add(BorderLayout.CENTER, mainPanel);

for (int i = 0; i < 256; i++) {
    JCheckBox c = new JCheckBox();
    c.setSelected(false);
    checkboxList.add(c);
    mainPanel.add(c);
} // fim do loop

theFrame.setBounds(50,50,300,300);
theFrame.pack();
theFrame.setVisible(true);
} // fecha buildGUI

public void setUpMidi() {
    try {
        sequencer = MidiSystem.getSequencer();
        sequencer.open();
        sequence = new Sequence(Sequence.PPQ,4);
        track = sequence.createTrack();
        sequencer.setTempoInBPM(120);
    } catch (Exception e) {e.printStackTrace();}
} // fecha setUpMidi

```

*JList é um componente que ainda não usamos. É onde as mensagens recebidas serão exibidas. Só que em vez de um bate-papo comum em que você apenas EXAMINARIA as mensagens, nesse aplicativo poderá SELECIONAR uma mensagem na lista e carregar e reproduzir o padrão de batida anexo.*

*Não há mais nada nessa página que seja novidade.*

*Captura o objeto Sequencer, cria um objeto Sequence e um objeto Track.*

```

public void buildTrackAndStart() {
    ArrayList<Integer> trackList = null; // conterá os instrumentos de cada faixa
    sequence.deleteTrack(track);
    track = sequence.createTrack();

    for (int i = 0; i < 16; i++) {
        trackList = new ArrayList<Integer>();

        for (int j = 0; j < 16; j++) {
            JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
            if (jc.isSelected()) {
                int key = instruments[i];
                trackList.add(new Integer(key));
            } else {
                trackList.add(null); // porque esse espaço deve ficar vazio na faixa
            }
        } // fecha o loop interno
        makeTracks(trackList);
    } // fecha o loop externo
    track.add(makeEvent(192,9,1,0,15)); // para sempre percorrermos todas as 16 batidas
    try {
        sequencer.setSequence(sequence);
        sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
        sequencer.start();
        sequencer.setTempoInBPM(120);
    } catch (Exception e) {e.printStackTrace();}
} // fecha o método

public class MyStartListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        buildTrackAndStart();
    } // fecha actionPerformed
} // fecha a classe interna

public class MyStopListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        sequencer.stop();
    } // fecha actionPerformed
} // fecha a classe interna

public class MyUpTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * 1.03));
    } // fecha actionPerformed
} // fecha a classe interna

public class MyDownTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * .97));
    }
}

public class MySendListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        // cria uma arraylist somente com o ESTADO das caixas de seleção
        boolean[] checkboxState = new boolean[256];
        for (int i = 0; i < 256; i++) {
            JCheckBox check = (JCheckBox) checkboxList.get(i);
            if (check.isSelected()) {
                checkboxState[i] = true;
            }
        } // fecha o loop
        String messageToSend = null;
        try {
            out.writeObject(userName + nextNum++ + ": " + userMessage.getText());
            out.writeObject(checkboxState);
        } catch (Exception ex) {
            System.out.println("Sorry dude. Could not send it to the server.");
        }
        userMessage.setText("");
    } // fecha actionPerformed
} // fecha a classe interna

```

*Constrói uma faixa percorrendo as caixas de seleção para capturar seu estado e convertê-lo em um instrumento (e constrói seu MidiEvent). Isso é muito complexo, mas ficou EXATAMENTE como nos capítulos anteriores, portanto consulte essas Receitas de Código para ver a explicação completa novamente.*

*Os ouvintes da GUI. Exatamente iguais aos da versão do capítulo anterior.*

*Isso é novo... Parece muito com SimpleChatClient, porém em vez de enviar uma mensagem em String, serializamos dois objetos (a mensagem em String e o padrão da batida) e os gravamos no fluxo de saída do soquete (para o servidor).*

```

public class MyListSelectionListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent le) {
        if (!le.getValueIsAdjusting()) {
            String selected = (String) incomingList.getSelectedValue();
            if (selected != null) {
                // agora vai até o mapa e altera a sequência
                boolean[] selectedState = (boolean[]) otherSeqsMap.get(selected);
                changeSequence(selectedState);
                sequencer.stop();
                buildTrackAndStart();
            }
        }
    } // fecha valueChanged
} // fecha a classe interna

public class RemoteReader implements Runnable {
    boolean[] checkboxState = null;
    String nameToShow = null;
    Object obj = null;
    public void run() {
        try {
            while((obj=in.readObject()) != null) {
                System.out.println("got an object from server");
                System.out.println(obj.getClass());
                String nameToShow = (String) obj;
                checkboxState = (boolean[]) in.readObject();
                otherSeqsMap.put(nameToShow, checkboxState);
                listVector.add(nameToShow);
                incomingList.setListData(listVector);
            } // fecha while
        } catch(Exception ex) {ex.printStackTrace();}
    } // fecha run
} // fecha a classe interna

public class MyPlayMineListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        if (mySequence != null) {
            sequence = mySequence; // restaura minha sequência original
        }
    } // fecha actionPerformed
} // fecha a classe interna

public void changeSequence(boolean[] checkboxState) {
    for (int i = 0; i < 256; i++) {
        JCheckBox check = (JCheckBox) checkboxList.get(i);
        if (checkboxState[i]) {
            check.setSelected(true);
        } else {
            check.setSelected(false);
        }
    } // fecha o loop
} // fecha changeSequence

public void makeTracks(ArrayList list) {
    Iterator it = list.iterator();
    for (int i = 0; i < 16; i++) {
        Integer num = (Integer) it.next();
        if (num != null) {
            int numKey = num.intValue();
            track.add(makeEvent(144,9,numKey, 100, i));
            track.add(makeEvent(128,9,numKey,100, i + 1));
        }
    } // fecha o loop
} // fecha makeTracks()

public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
        ShortMessage a = new ShortMessage();
        a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);
    } catch(Exception e) {}
    return event;
} // fecha makeEvent

} // fecha class

```

Isso também é novo - um ListSelectionListener que nos informará quando o usuário fizer uma seleção na lista de mensagens. Quando o usuário selecionar uma mensagem, carregaremos IMEDIATAMENTE o padrão de batida associado (ele estará no objeto HashMap chamado otherSeqsMap) e iniciaremos sua reprodução. Há alguns testes if por causa de pequenos problemas que ocorrem na captura de ListSelectionEvents.

Essa é a tarefa do segmento - ler dados no servidor. Nesse código, os 'dados' serão sempre dois objetos serializados: a mensagem em String e o padrão da batida (uma ArrayList com os valores do estado das caixas de seleção)

Quando uma mensagem chegar, leremos (desserializaremos) os dois objetos (a mensagem e a ArrayList com os valores booleanos do estado das caixas de seleção) e a adicionaremos ao componente JList. Incluir algo em uma JList é uma operação de duas etapas: você criará um objeto Vector com os dados das listas (Vector é uma ArrayList antiga) e, em seguida, informará à JList para usar esse objeto como a origem do que será exibido na lista.

Esse método será chamado quando o usuário selecionar algo na lista. Alteraremos IMEDIATAMENTE o padrão para o selecionado.

Todo o código relativo ao MIDI ficou exatamente igual ao da versão anterior.

Nada de novo. Exatamente como na última versão.

**Aponte seu lápis**

Quais seriam algumas das maneiras pelas quais você poderia aperfeiçoar esse programa?

Aqui estão algumas idéias para começar:

1) Quando você selecionar um padrão, o que estava sendo reproduzido será eliminado. Se for um padrão novo em que você estava trabalhando (ou uma alteração em outro padrão), isso não será bom. Talvez queira inserir uma caixa de diálogo que pergunte ao usuário se ele gostaria de salvar o padrão atual.

2) Se você não inserir um argumento de linha de comando, verá uma exceção quando executar o programa! Insira algo no método main que verifique se você passou um argumento de linha de comando. Se o usuário não fornecer um, use um padrão ou exiba uma mensagem que diga que eles precisam executar o programa novamente, mas dessa vez com um argumento com o nome de sua tela.

3) Pode ser bom ter um recurso em que você possa clicar em um botão e ele gere um padrão aleatório. Pode surgir um que você realmente goste. Melhor ainda, tenha outro recurso que lhe permita carregar padrões 'básicos' já existentes, como um para jazz, rock, reggae, etc., aos quais o usuário possa adicionar o que quiser.

Você pode encontrar os padrões já existentes no Web Start do livro Use a Cabeça! Java.

## Programa final do servidor da BeatBox

Grande parte desse código é idêntica ao do SimpleChatServer que criamos no capítulo sobre rede e segmentos. A única diferença, na verdade, é que esse servidor recebe e, em seguida, re-envia, dois objetos serializados em vez de uma String comum (embora um dos objetos serializados seja uma String).

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

public class MusicServer {

    ArrayList<ObjectOutputStream> clientOutputStreams;

    public static void main (String[] args) {
        new MusicServer().go();
    }

    public class ClientHandler implements Runnable {

        ObjectInputStream in;
        Socket clientSocket;

        public ClientHandler(Socket socket) {
            try {
                clientSocket = socket;
                in = new ObjectInputStream(clientSocket.getInputStream());

            } catch (Exception ex) {ex.printStackTrace();}
        } // fecha o construtor
    }
}
```

```

public void run() {
    Object o2 = null;
    Object o1 = null;
    try {

        while ((o1 = in.readObject()) != null) {

            o2 = in.readObject();

            System.out.println("read two objects");
            tellEveryone(o1, o2);
        } // fecha while

    } catch (Exception ex) {ex.printStackTrace();}
    } // fecha run
} // fecha a classe interna

public void go() {
    clientOutputStreams = new ArrayList<ObjectOutputStream>();

    try {
        ServerSocket serverSock = new ServerSocket(4242);

        while(true) {
            Socket clientSocket = serverSock.accept();
            ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
            clientOutputStreams.add(out);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();

            System.out.println("got a connection");
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
} // fecha go

public void tellEveryone(Object one, Object two) {
    Iterator it = clientOutputStreams.iterator();
    while(it.hasNext()) {
        try {
            ObjectOutputStream out = (ObjectOutputStream) it.next();
            out.writeObject(one);
            out.writeObject(two);
        } catch (Exception ex) {ex.printStackTrace();}
    }
} // fecha tellEveryone

} // close class

```