```
package appA;

import javax.sound.midi.*;
import javax.swing.*;
import javax.swipackage appA;

import javax.sound.midi.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import java.util.*;
import java.util.List;
import java.util.concurrent.*;

public class BeatBoxAlternative {
 private static final int NUMBER_OF_BEATS = 16;
 private final List<BeatInstrument> instruments;

 private final Map<BeatInstrument, List<JCheckBox>> instrumentCheckboxes = new
TreeMap<>();
 private JList<String> messages;
 private JTextField userMessage;

 private int nextNum;
 private String userName;
 private final Vector<String> incomingMessages = new Vector<>();
 private ObjectOutputStream out;
 private ObjectInputStream in;
 private final HashMap<String, boolean[]> otherSeqsMap = new HashMap<>();

 private Sequencer sequencer;
 private Sequence sequence;
 private Track track;

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

 public BeatBoxAlternative() {
```

```
  instruments = List.of(
          new BeatInstrument("Bass Drum", 35),
          new BeatInstrument("Closed Hi-Hat", 42),
          new BeatInstrument("Open Hi-Hat", 46),
          new BeatInstrument("Acoustic Snare", 38),
          new BeatInstrument("Crash Cymbal", 49),
          new BeatInstrument("Hand Clap", 39),
          new BeatInstrument("High Tom", 50),
          new BeatInstrument("Hi Bongo", 60),
          new BeatInstrument("Maracas", 70),
          new BeatInstrument("Whistle", 72),
          new BeatInstrument("Low Conga", 64),
          new BeatInstrument("Cowbell", 56),
          new BeatInstrument("Vibraslap", 58),
          new BeatInstrument("Low-mid Tom", 47),
          new BeatInstrument("High Agogo", 67),
          new BeatInstrument("Open Hi Conga", 63));
}

public void startUp(String name) {
  userName = name;
  try {
    Socket socket = new Socket("127.0.0.1", 4242);
    out = new ObjectOutputStream(socket.getOutputStream());
    in = new ObjectInputStream(socket.getInputStream());
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(new RemoteReader());
  } catch (IOException ex) {
    System.out.println("Couldn't connect-you'll have to play alone.");
  }
  setUpMidi();
  buildGUI();
}

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

  Box buttonBox = new Box(BoxLayout.Y_AXIS);
  JButton start = new JButton("Start");
```

```
start.addActionListener(e -> buildTrackAndStart());
buttonBox.add(start);

JButton stop = new JButton("Stop");
stop.addActionListener(e -> sequencer.stop());
buttonBox.add(stop);

JButton upTempo = new JButton("Tempo Up");
upTempo.addActionListener(e -> changeTempo(1.03));
buttonBox.add(upTempo);

JButton downTempo = new JButton("Tempo Down");
downTempo.addActionListener(e -> changeTempo(0.97));
buttonBox.add(downTempo);

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

userMessage = new JTextField();

buttonBox.add(userMessage);

messages = new JList<>();
messages.addListSelectionListener(new MyListSelectionListener());
messages.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
buttonBox.add(new JScrollPane(messages));
messages.setListData(incomingMessages); // no data to start with

Box nameBox = new Box(BoxLayout.Y_AXIS);
GridLayout grid = new GridLayout(instruments.size(), NUMBER_OF_BEATS, 2, 1);
JPanel mainPanel = new JPanel(grid);

for (BeatInstrument instrument : instruments) {
  JLabel instrumentName = new JLabel(instrument.getInstrumentName());
  instrumentName.setBorder(BorderFactory.createEmptyBorder(4, 1, 4, 1));
  nameBox.add(instrumentName);

  List<JCheckBox> checkboxList = new ArrayList<>();
  for (int i = 0; i < NUMBER_OF_BEATS; i++) {
    JCheckBox c = new JCheckBox();
    c.setSelected(false);
```

```
      checkboxList.add(c);
      mainPanel.add(c);
    }
    instrumentCheckboxes.put(instrument, checkboxList);
  }

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

  theFrame.getContentPane().add(background);
  theFrame.setBounds(50, 50, 300, 300);
  theFrame.pack();
  theFrame.setVisible(true);
 }

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

 public void buildTrackAndStart() {
  sequence.deleteTrack(track);
  track = sequence.createTrack();

  for (Map.Entry<BeatInstrument, List<JCheckBox>> instrumentsToBeats :
instrumentCheckboxes.entrySet()) {
    List<JCheckBox> checkboxes = instrumentsToBeats.getValue();
    for (int i = 0; i < checkboxes.size(); i++) {
      if (checkboxes.get(i).isSelected()) {
        BeatInstrument instrument = instrumentsToBeats.getKey();
        track.add(makeEvent(ShortMessage.NOTE_ON, instrument.getMidiValue(), 100,
i));
        track.add(makeEvent(ShortMessage.NOTE_OFF, instrument.getMidiValue(), 100, i
+ 1));
```

```
      }
    }
  }
  track.add(makeEvent(ShortMessage.PROGRAM_CHANGE, 1, 0, 15)); // - so we always go
to full 16 beats
  try {
    sequencer.setSequence(sequence);
    sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
    sequencer.start();
    sequencer.setTempoInBPM(120);
  } catch (InvalidMidiDataException e) {
    e.printStackTrace();
  }
}

public class MySendListener implements ActionListener {
  public void actionPerformed(ActionEvent a) {
    // make an array of just the STATE of the checkboxes
    boolean[] checkboxState = new boolean[256];

    int i = 0;
    for (List<JCheckBox> instrumentCheckboxes : instrumentCheckboxes.values()) {
      for (JCheckBox instrumentCheckbox : instrumentCheckboxes) {
        checkboxState[i++] = instrumentCheckbox.isSelected();
      }
    }
    try {
      out.writeObject(userName + nextNum++ + ": " + userMessage.getText());
      out.writeObject(checkboxState);
    } catch (IOException ex) {
      System.out.println("Terribly sorry. Could not send it to the server.");
      ex.printStackTrace();
    }
    userMessage.setText("");
  }
}

public class MyListSelectionListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent le) {
    if (!le.getValueIsAdjusting()) {
      String selected = messages.getSelectedValue();
      if (selected != null) {
```

```
        // now go to the map, and change the sequence
        boolean[] selectedState = otherSeqsMap.get(selected);
        changeSequence(selectedState);
        sequencer.stop();
        buildTrackAndStart();
      }
    }
  }
}

public class RemoteReader implements Runnable {
  public void run() {
    try {
      Object obj;
      while ((obj = in.readObject()) != null) {
        System.out.println("got an object from server");
        String nameToShow = (String) obj;
        boolean[] checkboxState = (boolean[]) in.readObject();
        otherSeqsMap.put(nameToShow, checkboxState);
        incomingMessages.add(nameToShow);
        messages.setListData(incomingMessages);
      }
    } catch (IOException | ClassNotFoundException ex) {
      ex.printStackTrace();
    }
  }
}

private void changeSequence(boolean[] newCheckboxStates) {
  int i = 0;
  for (List<JCheckBox> checkboxesForInstrument : instrumentCheckboxes.values()) {
    for (JCheckBox checkbox : checkboxesForInstrument) {
      checkbox.setSelected(newCheckboxStates[i++]);
    }
  }
}

private MidiEvent makeEvent(int command, int one, int two, int tick) {
  try {
    ShortMessage midiMessage = new ShortMessage(command, 9, one, two);
    return new MidiEvent(midiMessage, tick);
  } catch (InvalidMidiDataException ignored) {
```

```
      return null;
    }
  }

  private void changeTempo(double tempoMultiplier) {
    float tempoFactor = sequencer.getTempoFactor();
    sequencer.setTempoFactor((float) (tempoFactor * tempoMultiplier));
  }

  private static final class BeatInstrument implements Comparable<BeatInstrument> {
    private final String instrumentName;
    private final int midiValue;

    BeatInstrument(String instrumentName, int midiValue) {
      this.instrumentName = instrumentName;
      this.midiValue = midiValue;
    }

    public String getInstrumentName() {
      return instrumentName;
    }

    public int getMidiValue() {
      return midiValue;
    }

    public int compareTo(BeatInstrument other) {
      return instrumentName.compareTo(other.instrumentName);
    }
  }
}
```

```
ng.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import java.util.*;
import java.util.List;
import java.util.concurrent.*;

public class BeatBoxAlternative {
 private static final int NUMBER_OF_BEATS = 16;
 private final List<BeatInstrument> instruments;

 private final Map<BeatInstrument, List<JCheckBox>> instrumentCheckboxes = new
TreeMap<>();
 private JList<String> messages;
 private JTextField userMessage;

 private int nextNum;
 private String userName;
 private final Vector<String> incomingMessages = new Vector<>();
 private ObjectOutputStream out;
 private ObjectInputStream in;
 private final HashMap<String, boolean[]> otherSeqsMap = new HashMap<>();

 private Sequencer sequencer;
 private Sequence sequence;
 private Track track;

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

 public BeatBoxAlternative() {
   instruments = List.of(
           new BeatInstrument("Bass Drum", 35),
           new BeatInstrument("Closed Hi-Hat", 42),
           new BeatInstrument("Open Hi-Hat", 46),
           new BeatInstrument("Acoustic Snare", 38),
           new BeatInstrument("Crash Cymbal", 49),
           new BeatInstrument("Hand Clap", 39),
           new BeatInstrument("High Tom", 50),
```

```
          new BeatInstrument("Hi Bongo", 60),
          new BeatInstrument("Maracas", 70),
          new BeatInstrument("Whistle", 72),
          new BeatInstrument("Low Conga", 64),
          new BeatInstrument("Cowbell", 56),
          new BeatInstrument("Vibraslap", 58),
          new BeatInstrument("Low-mid Tom", 47),
          new BeatInstrument("High Agogo", 67),
          new BeatInstrument("Open Hi Conga", 63));
}

public void startUp(String name) {
  userName = name;
  try {
    Socket socket = new Socket("127.0.0.1", 4242);
    out = new ObjectOutputStream(socket.getOutputStream());
    in = new ObjectInputStream(socket.getInputStream());
    ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.submit(new RemoteReader());
  } catch (IOException ex) {
    System.out.println("Couldn't connect-you'll have to play alone.");
  }
  setUpMidi();
  buildGUI();
}

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

  Box buttonBox = new Box(BoxLayout.Y_AXIS);
  JButton start = new JButton("Start");
  start.addActionListener(e -> buildTrackAndStart());
  buttonBox.add(start);

  JButton stop = new JButton("Stop");
  stop.addActionListener(e -> sequencer.stop());
  buttonBox.add(stop);

  JButton upTempo = new JButton("Tempo Up");
```

```
upTempo.addActionListener(e -> changeTempo(1.03));
buttonBox.add(upTempo);

JButton downTempo = new JButton("Tempo Down");
downTempo.addActionListener(e -> changeTempo(0.97));
buttonBox.add(downTempo);

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

userMessage = new JTextField();

buttonBox.add(userMessage);

messages = new JList<>();
messages.addListSelectionListener(new MyListSelectionListener());
messages.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
buttonBox.add(new JScrollPane(messages));
messages.setListData(incomingMessages); // no data to start with

Box nameBox = new Box(BoxLayout.Y_AXIS);
GridLayout grid = new GridLayout(instruments.size(), NUMBER_OF_BEATS, 2, 1);
JPanel mainPanel = new JPanel(grid);

for (BeatInstrument instrument : instruments) {
  JLabel instrumentName = new JLabel(instrument.getInstrumentName());
  instrumentName.setBorder(BorderFactory.createEmptyBorder(4, 1, 4, 1));
  nameBox.add(instrumentName);

  List<JCheckBox> checkboxList = new ArrayList<>();
  for (int i = 0; i < NUMBER_OF_BEATS; i++) {
    JCheckBox c = new JCheckBox();
    c.setSelected(false);
    checkboxList.add(c);
    mainPanel.add(c);
  }
  instrumentCheckboxes.put(instrument, checkboxList);
}

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

```
    background.add(BorderLayout.CENTER, mainPanel);

    theFrame.getContentPane().add(background);
    theFrame.setBounds(50, 50, 300, 300);
    theFrame.pack();
    theFrame.setVisible(true);
  }

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

  public void buildTrackAndStart() {
    sequence.deleteTrack(track);
    track = sequence.createTrack();

    for (Map.Entry<BeatInstrument, List<JCheckBox>> instrumentsToBeats :
instrumentCheckboxes.entrySet()) {
      List<JCheckBox> checkboxes = instrumentsToBeats.getValue();
      for (int i = 0; i < checkboxes.size(); i++) {
        if (checkboxes.get(i).isSelected()) {
          BeatInstrument instrument = instrumentsToBeats.getKey();
          track.add(makeEvent(ShortMessage.NOTE_ON, instrument.getMidiValue(), 100,
i));
          track.add(makeEvent(ShortMessage.NOTE_OFF, instrument.getMidiValue(), 100, i
+ 1));
        }
      }
    }
    track.add(makeEvent(ShortMessage.PROGRAM_CHANGE, 1, 0, 15)); // - so we always go
to full 16 beats
    try {
      sequencer.setSequence(sequence);
      sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
```

```
    sequencer.start();
    sequencer.setTempoInBPM(120);
  } catch (InvalidMidiDataException e) {
    e.printStackTrace();
  }
}

public class MySendListener implements ActionListener {
  public void actionPerformed(ActionEvent a) {
    // make an array of just the STATE of the checkboxes
    boolean[] checkboxState = new boolean[256];

    int i = 0;
    for (List<JCheckBox> instrumentCheckboxes : instrumentCheckboxes.values()) {
      for (JCheckBox instrumentCheckbox : instrumentCheckboxes) {
        checkboxState[i++] = instrumentCheckbox.isSelected();
      }
    }
    try {
      out.writeObject(userName + nextNum++ + ": " + userMessage.getText());
      out.writeObject(checkboxState);
    } catch (IOException ex) {
      System.out.println("Terribly sorry. Could not send it to the server.");
      ex.printStackTrace();
    }
    userMessage.setText("");
  }
}

public class MyListSelectionListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent le) {
    if (!le.getValueIsAdjusting()) {
      String selected = messages.getSelectedValue();
      if (selected != null) {
        // now go to the map, and change the sequence
        boolean[] selectedState = otherSeqsMap.get(selected);
        changeSequence(selectedState);
        sequencer.stop();
        buildTrackAndStart();
      }
    }
  }
```

```
}

public class RemoteReader implements Runnable {
  public void run() {
    try {
      Object obj;
      while ((obj = in.readObject()) != null) {
        System.out.println("got an object from server");
        String nameToShow = (String) obj;
        boolean[] checkboxState = (boolean[]) in.readObject();
        otherSeqsMap.put(nameToShow, checkboxState);
        incomingMessages.add(nameToShow);
        messages.setListData(incomingMessages);
      }
    } catch (IOException | ClassNotFoundException ex) {
      ex.printStackTrace();
    }
  }
}

private void changeSequence(boolean[] newCheckboxStates) {
  int i = 0;
  for (List<JCheckBox> checkboxesForInstrument : instrumentCheckboxes.values()) {
    for (JCheckBox checkbox : checkboxesForInstrument) {
      checkbox.setSelected(newCheckboxStates[i++]);
    }
  }
}

private MidiEvent makeEvent(int command, int one, int two, int tick) {
  try {
    ShortMessage midiMessage = new ShortMessage(command, 9, one, two);
    return new MidiEvent(midiMessage, tick);
  } catch (InvalidMidiDataException ignored) {
    return null;
  }
}

private void changeTempo(double tempoMultiplier) {
  float tempoFactor = sequencer.getTempoFactor();
  sequencer.setTempoFactor((float) (tempoFactor * tempoMultiplier));
}
```

```
 private static final class BeatInstrument implements Comparable<BeatInstrument> {
   private final String instrumentName;
   private final int midiValue;

   BeatInstrument(String instrumentName, int midiValue) {
     this.instrumentName = instrumentName;
     this.midiValue = midiValue;
   }

   public String getInstrumentName() {
     return instrumentName;
   }

   public int getMidiValue() {
     return midiValue;
   }

   public int compareTo(BeatInstrument other) {
     return instrumentName.compareTo(other.instrumentName);
   }
 }
}
```

```
package appA;
```

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

public class MusicServer {
 private final List<ObjectOutputStream> clientOutputStreams = new ArrayList<>();

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

 public void go() {
   try {
     ServerSocket serverSock = new ServerSocket(4242);
     ExecutorService threadPool = Executors.newCachedThreadPool();

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

       ClientHandler clientHandler = new ClientHandler(clientSocket);
       threadPool.execute(clientHandler);
       System.out.println("Got a connection");
     }
   } catch (IOException e) {
     e.printStackTrace();
   }
 }

 public void tellEveryone(Object usernameAndMessage, Object beatSequence) {
   for (ObjectOutputStream clientOutputStream : clientOutputStreams) {
     try {
       clientOutputStream.writeObject(usernameAndMessage);
       clientOutputStream.writeObject(beatSequence);
     } catch (IOException e) {
       e.printStackTrace();
     }
   }
 }
```

```
public class ClientHandler implements Runnable {
  private ObjectInputStream in;

  public ClientHandler(Socket socket) {
    try {
      in = new ObjectInputStream(socket.getInputStream());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  public void run() {
    Object userNameAndMessage;
    Object beatSequence;
    try {
      while ((userNameAndMessage = in.readObject()) != null) {
        beatSequence = in.readObject();

        System.out.println("read two objects");
        tellEveryone(userNameAndMessage, beatSequence);
      }
    } catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}

}
```