```
1  package jms.test.fetch;
2
3  import javax.jms.JMSException;
4  import javax.jms.Message;
5  import javax.jms.Queue;
6  import javax.jms.QueueConnection;
7  import javax.jms.QueueConnectionFactory;
8  import javax.jms.QueueReceiver;
9  import javax.jms.QueueSession;
10 import javax.jms.Session;
11 import javax.jms.TextMessage;
12
13 import org.mr.api.jms.MantaQueueConnectionFactory;
14
15 public class Receiver
16 {
17     private String myName;
18     private QueueConnection con;
19     private QueueReceiver receiver;
20
21     public Receiver(String myName) throws JMSException
22     {
23         this.myName = myName;
24
25         // create a connection object via a factory:
26         QueueConnectionFactory conFactory =
27             (QueueConnectionFactory) new MantaQueueConnectionFactory();
28         con = conFactory.createQueueConnection();
29
30         // create a queue and an associated receiver:
31         QueueSession session =
32             (QueueSession) con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
33         Queue receiveQueue = session.createQueue(myName);
34         receiver = session.createReceiver(receiveQueue);
35
36         // enable messaging to start:
37         con.start();
38     }
39
40     public void fetchMessages() throws JMSException
41     {
42         while (true)
43         {
44             Message msg = receiver.receive();
45
46             if (msg instanceof TextMessage)
47             {
48                 TextMessage tmsg = (TextMessage) msg;
49                 System.out.println(myName +
50                         ": received: " + tmsg.getText());
51             }
52         }
53     }
54
55     public static void main(String[] args) throws JMSException
56     {
57         Receiver r = new Receiver("receiver");
58         r.fetchMessages();
59     }
60 }
```

Figure 2.2: A simple JMS receiver program.

```java
package jms.test.listen;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
... etc

import org.mr.api.jms.MantaQueueConnectionFactory;

public class Receiver implements MessageListener
{
    private String myName;
    private QueueConnection con;

    public Receiver(String myName) throws JMSException
    {
        this.myName = myName;

        // create a connection object via a factory:
        QueueConnectionFactory conFactory =
            (QueueConnectionFactory) new MantaQueueConnectionFactory();
        con = conFactory.createQueueConnection();

        // create a queue and a receiver to listen on:
        QueueSession session =
            (QueueSession) con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue receiveQueue = session.createQueue(myName);
        QueueReceiver receiver = session.createReceiver(receiveQueue);

        // attach itself as a listener to the queue:
        receiver.setMessageListener(this);

        // enable messaging to start:
        con.start();
    }

    // this method will be called whenever a message arrives:
    @Override
    public void onMessage(Message msg)
    {
        if (msg instanceof TextMessage)
        {
            TextMessage tmsg = (TextMessage) msg;
            try
            {
                System.out.println(myName + ": received: " + tmsg.getText());
            }
            catch (JMSException e)
            {
                throw new Error(myName + ": could not extract text from message: "
                + e.getMessage());
            }
        }
    }

    public static void main(String[] args) throws JMSException
    {
        new Receiver("receiver");
    }
}
```

Figure 2.3: A simple JMS receiver program **listening** to the queue.

```java
package jms.test.fetch;

import javax.jms.JMSException;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.mr.api.jms.MantaQueueConnectionFactory;

public class Sender
{
    private String myName;
    private QueueConnection con;
    private QueueSession session;

    public Sender(String myName) throws JMSException
    {
        this.myName = myName;

        // create a connection object via a factory:
        QueueConnectionFactory conFactory =
            (QueueConnectionFactory) new MantaQueueConnectionFactory();
        con = conFactory.createQueueConnection();

        // create a session for sending messages:
        session =
            (QueueSession) con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

        // enable messaging to start:
        con.start();
    }

    public void sendMessage(String destination, String text) throws JMSException
    {
        Queue sendQueue = session.createQueue(destination);
        QueueSender sender = session.createSender(sendQueue);
        TextMessage tmsg = session.createTextMessage(myName + " says: " + text);
        sender.send(tmsg);
    }

    public static void main(String[] args) throws JMSException
    {
        Sender s = new Sender("sender");
        s.sendMessage("receiver", "hello!");
    }
}
```

Figure 2.4: A simple JMS sender program.

```
1  public class Dispatcher implements javax.jms.MessageListener
2  {
3    ...
4      public void onMessage(Message msg)
5      {
6          // pull out an object from the message:
7          ObjectMessage omsg = (ObjectMessage) msg;
8          Serializable obj;
9          try
10         {
11             obj = omsg.getObject();
12         }
13         catch (JMSException e)
14         {
15             throw new Error("wisdom.peer.Dispatcher.onMessage: "
16                     + "failed to retrieve message object: " + e.getMessage());
17         }
18         ...
19         else if (obj instanceof NewWisdom)
20         {
21             NewWisdom nw = (NewWisdom) obj;
22
23             if (wisdomIDs.add(nw.getWisdom().getId())) // test if new
24             {
25                 System.out.println("received new wisdom: " + nw);
26                 processNewWisdom(nw);
27             }
28         }
29         ...
30     }
31     private void processNewWisdom(NewWisdom nw)
32     {
33         // store this wisdom:
34         ...
35         // forward the whole wisdom to the neighbours:
36         forwardWisdom(nw);
37     }
38     private void forwardWisdom(NewWisdom nw)
39     {
40         try { neighbours.newWisdom(nw); } catch ...
41     }
42   ...
43     private Neighbours neighbours;
44 }
45
46 public class Neighbours
47 {
48   ...
49     public synchronized void newWisdom(NewWisdom nw) throws JMSException
50     {
51         ...
52         ObjectMessage msg = sendSession.createObjectMessage(nw);
53
54         for (QueueSender neighbour : neighbours.values())
55         {
56             neighbour.send(msg, DeliveryMode.NON_PERSISTENT,
57                     Message.DEFAULT_PRIORITY, MESSAGE_TTL);
58         }
59     }
60   ...
61 }
```

Figure 2.6: Simple boadcast routing in a P2P chat system.