

developers.sun.com
» [search tips](#) |Search: in [Developers' Site](#) ▼

The Source for Developers
A Sun Developer Network Site

» [Products & Technologies](#)
» [Technical Topics](#)



[Developers Home](#) > [Products & Technologies](#) > [Java Technology](#) > [J2SE](#) > [Reference](#) > [Technical Articles and Tips](#) >

[Join a Sun Developer Network Community](#)
[Profile and Registration](#) | [Why Register?](#)

Technical Articles and Tips

Sorting Lists and Sending Mail With the JavaMail API

[Print-friendly Vers](#)
[Tech Tips Archive](#)
October 23, 2001

WELCOME to the Java Developer Connection (JDC) Tech Tips, October 23, 2001. This issue covers:

- n [Sorting Lists](#)
- n [Sending Mail With the JavaMail API](#)

This issue of the JDC Tech Tips is written by John Zukowski, president of [JZ Ventures, Inc.](#)

These tips were developed using Java 2 SDK, Standard Edition, v 1.3.

SORTING LISTS

The `List` implementations found in the Java Collections Framework are `Vector`, `ArrayList`, and `LinkedList`. These collections offer indexed access to groups of objects. They provide support for adding and removing elements. However they offer no built-in support for sorting elements.

To sort `List` elements you can use the `sort()` method in the `java.util.Collections` class. You can either pass the method a `List` object, or you can pass it a `List` and a `Comparator`. If the elements in the `List` are all the same class type and that class implements the `Comparable` interface, you can simply call `Collections.sort()`. If, however, the class doesn't implement `Comparator`, you can pass the `sort()` method a `Comparator` to do the sort. You might also want to pass the `sort()` method a `Comparator` if you don't want to use the default sort order for the elements of the class. If the elements in the `List` are not all the same class type, you're out of luck as far as sorting goes -- unless you write a special cross-class `Comparator`.

What about sorting order? If the elements are `String` objects, the default sort order is based on the character codes, basically the ASCII/Unicode value for each character. If you strictly work with English, the default sort order is usually sufficient because it will sort A-Z first, followed by the lowercase letters a-z. However, if you work with non-English words, or you just want a different sorting order, that's where the second variety of `Collections.sort()` comes in handy. For instance, say you want to sort the elements of the `List` in the reverse natural order of the strings. To do this, you can get a reverse `Comparator` from the `Collections` class with `reverseOrder()`. Then, you pass the reverse `Comparator` to the `sort()` method. In other words, you do the following:

```
List list = ...;
Comparator comp = Collections.reverseOrder();
Collections.sort(list, comp);
```

If the list contained the terms Man, man, Woman, and woman, the sorted list would be Man, Woman, man, woman. Nothing too complicated here. An important thing to remember is `Collections.sort()` does an in-place sort. If you need to retain the original order, make a copy of the collection first, then sort it, like this:

```
List list = ...;
List copyOfList = new ArrayList(list);
Collections.sort(copyOfList);
```

Here, the sorted list is Man, Woman, man, woman, but the original list (Man, man, Woman, and woman) is retained.

So far the sorting has been case-sensitive. How would you do a case-insensitive sort? One way to do it is to implement a `Comparator` like this

```
public static class CaseInsensitiveComparator
    implements Comparator {
    public int compare(Object element1,
        Object element2) {
        String lower1 =
            element1.toString().toLowerCase();
        String lower2 =
            element2.toString().toLowerCase();
        return lower1.compareTo(lower2);
    }
}
```

You really don't have to manually create this class. Instead, you can use the preexisting `Comparator`, `CASE_INSENSITIVE_ORDER` found in the `String` class.

There is a slight problem with this approach. The `sort()` algorithm provides stable sorting, keeping equal elements in the original order. This means that a list that contains the two elements "woman" and "Woman" would sort differently based upon which of the two elements appear first in the list.

What about language differences? The `java.text` package provides `Collator` and `CollationKey` classes for doing language-sensitive sorting. Here's an example:

```
public static class CollatorComparator
    implements Comparator {
    Collator collator = Collator.getInstance();
    public int compare(Object element1,
        Object element2) {
        CollationKey key1 = collator.getCollationKey(
            element1.toString());
        CollationKey key2 = collator.getCollationKey(
            element2.toString());
        return key1.compareTo(key2);
    }
}
```

Note that if your text is for a locale other than the default, you need to pass that locale to the `getInstance()` method, as in the following:

```
Collator collator = Collator.getInstance(Locale.JAPANESE);
```

Instead of sorting on the actual string, you sort on a collation key. Not only does this provide consistent case-insensitive sorting, but it also works across languages. In other words, if you sort a combination of Spanish and non-Spanish words, the word *mañana* (tomorrow) would come before *mantra*. If you don't use a `Collator`, *mañana* would come after *mantra*.

Here's a program that does various types of sorting (default, case-insensitive, and language-sensitive) on a List:

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.io.*;
import java.text.*;
import java.util.*;
import javax.swing.*;

public class SortIt {

    public static class CollatorComparator
        implements Comparator {
        Collator collator = Collator.getInstance();
        public int compare(Object element1,
            Object element2) {
            CollationKey key1 = collator.getCollationKey(
                element1.toString());
            CollationKey key2 = collator.getCollationKey(
                element2.toString());
            return key1.compareTo(key2);
        }
    }

    public static class CaseInsensitiveComparator
        implements Comparator {
        public int compare(Object element1,
            Object element2) {
            String lower1 = element1.toString().
                toLowerCase();
            String lower2 = element2.toString().
                toLowerCase();
            return lower1.compareTo(lower2);
        }
    }

    public static void main(String args[]) {
        String words[] =
            {"man", "Man", "Woman", "woman",
             "Manana", "manana", "mañana", "Mañana",
             "Mantra", "mantra", "mantel", "Mantel"};

        // Create frame to display sortings
        JFrame frame = new JFrame("Sorting");
        frame.setDefaultCloseOperation(
```

```

.....\
    JFrame.EXIT_ON_CLOSE);
    Container contentPane = frame.getContentPane();
    JTextArea textArea = new JTextArea();
    JScrollPane pane = new JScrollPane(textArea);
    contentPane.add(pane, BorderLayout.CENTER);

    // Create buffer for output
    StringWriter buffer = new StringWriter();
    PrintWriter out = new PrintWriter(buffer);

    // Create initial list to sort
    List list = new ArrayList(Arrays.asList(words));
    out.println("Original list:");
    out.println(list);
    out.println();

    // Perform default sort
    Collections.sort(list);
    out.println("Default sorting:");
    out.println(list);
    out.println();

    // Reset list
    list = new ArrayList(Arrays.asList(words));

    // Perform case insensitive sort
    Comparator comp = new CaseInsensitiveComparator();
    Collections.sort(list, comp);
    out.println("Case insensitive sorting:");
    out.println(list);
    out.println();

    // Reset list
    list = new ArrayList(Arrays.asList(words));

    // Perform collation sort
    comp = new CollatorComparator();
    Collections.sort(list, comp);
    out.println("Collator sorting:");
    out.println(list);
    out.println();

    // Fill text area and display
    textArea.setText(buffer.toString());
    frame.pack();
    frame.show();
}
}

```

If your primary concern is ordered access, perhaps a `List` isn't the best data structure for you. As long as your collection doesn't have duplicates, you can keep the elements in a `TreeSet` (with or without providing a `Comparator`). Then, the elements will always be in sorted order.

For more information about sorting `Lists`, see "[Introduction to the Collections Framework](#)"

SENDING MAIL WITH THE JAVAMAIL API

The JavaMail API is a standard part of the Java 2 Platform, Enterprise Edition (J2EE). The API is downloadable from <http://java.sun.com/products/javamail/>. As its name suggests, the JavaMail API provides support for sending and receiving mail messages. You still must connect to your mail server for sending and receiving the messages because the JavaMail API is not a mail server, but the API does provide the framework for making that connection.

Sending mail involves understanding at least four classes:

```

Session
Message
Address
Transport

```

And, if you are interested in including an attachment with your message, you must know a few more:

```

Multipart
BodyPart
DataSource
DataHandler

```

The `DataSource` class and its implementations `FileDataSource` and `URLDataSource` are part of the JavaBeans Activation Framework. The `DataHandler` class is a part of that framework too. The JavaBeans Activation Framework is included with J2EE. The Framework is downloadable from <http://java.sun.com/products/javabeans/glasgow/jaf.html>. The JavaMail API requires the JavaBeans Activation Framework

library.

Looking at the specific classes in the JavaMail API, first there is the `Session` class. This represents the basic mail session with the mail server. All operations require connecting to a mail server, so you must create the connection. You get the session with either `getDefaultInstance()` or `getInstance()` methods of the `Session` class. The `getDefaultInstance()` method gets a shared session instance. By comparison, the `getInstance()` method gets a new session instance each time it's called. When getting the session, you must pass a `Properties` instance, this contains property settings such as the mail server to be used for the connection. In the following example, the `mail.smtp.host` property is passed to `Session.getDefaultInstance`. The property contains the name of the mail server for the connection (`smtp.example.com`):

```
Properties props = new Properties();
props.put("mail.smtp.host",
    "smtp.example.com");
Session session =
    Session.getDefaultInstance(props, null);
```

After you have the `Session`, you can create a `Message` from the `Session`. As its name implies, the `Message` class represents the email message. It is abstract, so you must create an instance of a subclass. That subclass is `MimeMessage`:

```
MimeMessage message = new MimeMessage(session);
```

Next, you need to fill the message's content. Each of the pieces of the message has a different method to set the content:

```
setFrom() is used to set the From address
addRecipient() is used to set the To, CC, and BCC addresses
setSubject() is used to set the Subject
setText() is used to set the content
```

In order to set the `From` or `To` fields of the message, you need an `Address`, which you create with the `InternetAddress` class. You create a `Address` from a string, and pass the `Address` to the appropriate method. You can provide just the email address or an address and a name string. For example:

```
Address address = new InternetAddress(
    "me@example.com");
Address address = new InternetAddress(
    "me@example.com", "Just Me");
```

To set the `From` field, simply pass the `Address` to `setFrom()`, that is, `setFrom(address)`.

The `addRecipient()` method involves a little more work than using `setFrom()`. With `addRecipient()`, you need to specify whether the `Address` is for the `To` field, the `CC` field, or the `BCC` (Blind carbon copy) field. This is done with the help of the `RecipientType` class:

```
Message.RecipientType.TO
Message.RecipientType.CC
Message.RecipientType.BCC
```

This means that if you want to set the `To` field, you provide `Message.RecipientType.TO` and the `Address` to the `addRecipient()` method:

```
message.addRecipient(Message.RecipientType.TO,
    address);
```

For a basic message, that's all you need to do. The `setSubject()` and `setText()` methods just accept the text content of their respective message pieces.

After your message is set, send it with the `Transport` class:

```
Transport.send(message);
```

This uses the SMTP server specified in the properties for the session.

Here's a complete program that uses the JavaMail API to send a message. To run the program, issue the command:

```
JDCSend SMTP_host from_address to_address
```

Replace `SMTP_host` with the name of an SMTP host, `from_address` with an address for the `From` field of the message, and `to_address` with an address for the `To` field of the message.

```

import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;

public class JDCSend {
    public static void main (String args[])
        throws Exception {
        String smtpHost = args[0];
        String from = args[1];
        String to = args[2];

        // Get system properties
        Properties props = System.getProperties();

        // Setup mail server
        props.put("mail.smtp.host", smtpHost);

        // Get session
        Session session =
            Session.getDefaultInstance(props, null);

        // Define message
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO,
            new InternetAddress(to));
        message.setSubject("Hello, JDC");
        message.setText("Welcome to the JDC");

        // Send message
        Transport.send(message);
    }
}

```

If you want to include an attachment with your message, you need to build up the parts, quite literally, because the name of the applicable interface is `Part`. The content of your `Message` will consist of multiple parts within a `Multipart` object. Part one of the message is a `BodyPart` that contains the message content. Part two of the message is a `BodyPart` that contains the attachment. The attachment itself is specified as a `DataSource`. You don't have to actually read the attachment.

You start in the same way as you do for a message without an attachment. Create the message from the session and initialize the headers:

```

Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
    new InternetAddress(to));
message.setSubject("JDC Attachment");

```

However here you need to create the `Multipart` object:

```
Multipart multipart = new MimeMultipart();
```

For part one, create a `BodyPart` and set the text to be a message. Then, add the `BodyPart` to the `Multipart` you just created.

```

BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText("Here's the file");
multipart.addBodyPart(messageBodyPart);

```

For part two, you need to create a `BodyPart` again, but this time you need to create a `DataSource` for the file.

```

messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(filename);

```

Use a `DataHandler` object to attach the data source to the message. Simply create a `DataHandler` for the source and attach it to the message:

```

messageBodyPart.setDataHandler(
    new DataHandler(source));

```

Remember to set the filename of the attachment. This permits the recipient to know the name (and type) of the received file.

```
messageBodyPart.setFileName(filename);
```

Attach part two in the same way as part one:

```
multipart.addBodyPart(messageBodyPart);
```

And as a final step before sending, attach the Multipart to the Message:

```
message.setContent(multipart);
```

You can now send the message. Here's a complete program. Make sure to pass the proper command-line arguments to the program, as follows:

```
JDCAttach SMTP_host from_address to_address attach_file
```

Replace SMTP_host with the name of an SMTP host, from_address with an address for the From field of the message, to_address with a used for the To field of the message, and attach_file with the filename of the attachment.

```
import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class JDCAttach {
    public static void main (String args[])
        throws Exception {
        String smtpHost = args[0];
        String from = args[1];
        String to = args[2];
        String filename = args[3];

        // Get system properties
        Properties props = System.getProperties();

        // Setup mail server
        props.put("mail.smtp.host", smtpHost);

        // Get session
        Session session = Session.getDefaultInstance(
            props, null);

        // Define message
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO,
            new InternetAddress(to));
        message.setSubject("Hello, JDC Attachment");

        // Create the multi-part
        Multipart multipart = new MimeMultipart();

        // Create part one
        BodyPart messageBodyPart = new MimeBodyPart();

        // Fill the message
        messageBodyPart.setText("Here's the file");

        // Add the first part
        multipart.addBodyPart(messageBodyPart);

        // Part two is attachment
        messageBodyPart = new MimeBodyPart();
        DataSource source = new FileDataSource(filename);
        messageBodyPart.setDataHandler(
            new DataHandler(source));
        messageBodyPart.setFileName(filename);

        // Add the second part
        multipart.addBodyPart(messageBodyPart);

        // Put parts in message
        message.setContent(multipart);

        // Send message
        Transport.send(message);
    }
}
```

For more information about the JavaMail API, see "[Fundamentals of the JavaMail API](#)". For a demonstration of the JavaMail API in use in a rea

application, see "[The Java Technologies Behind ICEMail: An Open-Source Project](#)"

IMPORTANT: Please read our Terms of Use and Privacy policies:

<http://www.sun.com/share/text/termsfuse.html>

<http://www.sun.com/privacy/>

FEEDBACK

Comments? Send your feedback on the JDC Tech Tips to:

jdc-webmaster@sun.com

SUBSCRIBE/UNSUBSCRIBE

- To subscribe, go to the [subscriptions page](#), choose the newsletters you want to subscribe to and click "Update".
- To unsubscribe, go to the [subscriptions page](#), uncheck the appropriate checkbox, and click "Update".

- ARCHIVES

You'll find the JDC Tech Tips archives at:

<http://java.sun.com/jdc/TechTips/index.html>

- COPYRIGHT

Copyright 2001 Sun Microsystems, Inc. All rights reserved.
901 San Antonio Road, Palo Alto, California 94303 USA.

This document is protected by copyright. For more information, see:

<http://java.sun.com/jdc/copyright.html>

JDC Tech Tips
October 23, 2001

Sun, Sun Microsystems, Java, Java Developer Connection, J2EE, and JavaMail are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.



[Company Info](#) | [About This Site](#) | [Press](#) | [Contact Us](#) | [Employment](#)
[How to Buy](#) | [Licensing](#) | [Terms of Use](#) | [Privacy](#) | [Trademarks](#)

Copyright 1994-2004 Sun Microsystems, Inc.

A Sun Developer Network Site

Unless otherwise licensed, code all technical manuals herein (including articles, FAQs, samples) is provided under this [License](#).

 [Content Feeds](#)