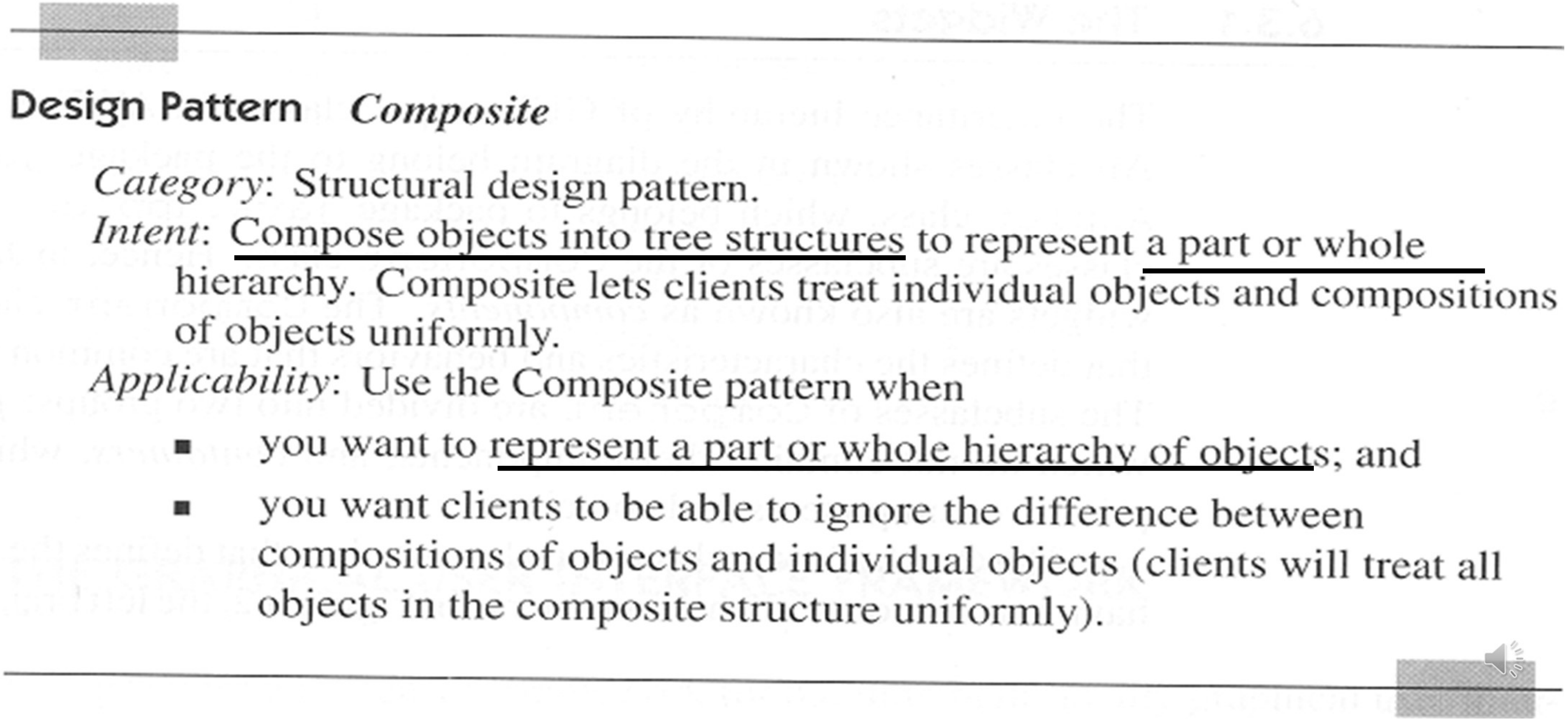
10.4.2 Design Pattern : Composite (page 532)

- The Composite design pattern
  - discussed in Section 8.3.2 [page 336]
  - used to build a hierarchical structure of objects
  - an example of such a hierarchy can be found in various e-mail programs
    - a mail-box consists of a number of mail folders
    - a mail-folder consist of a number of mails and/or sub-mail folders; that is , mail folders can be nested any number of levels

8.3.2 Design Pattern : Composite

- Ex) The design of the GUI components hierarchy

- Design Pattern: Composite



Design Pattern *Composite*

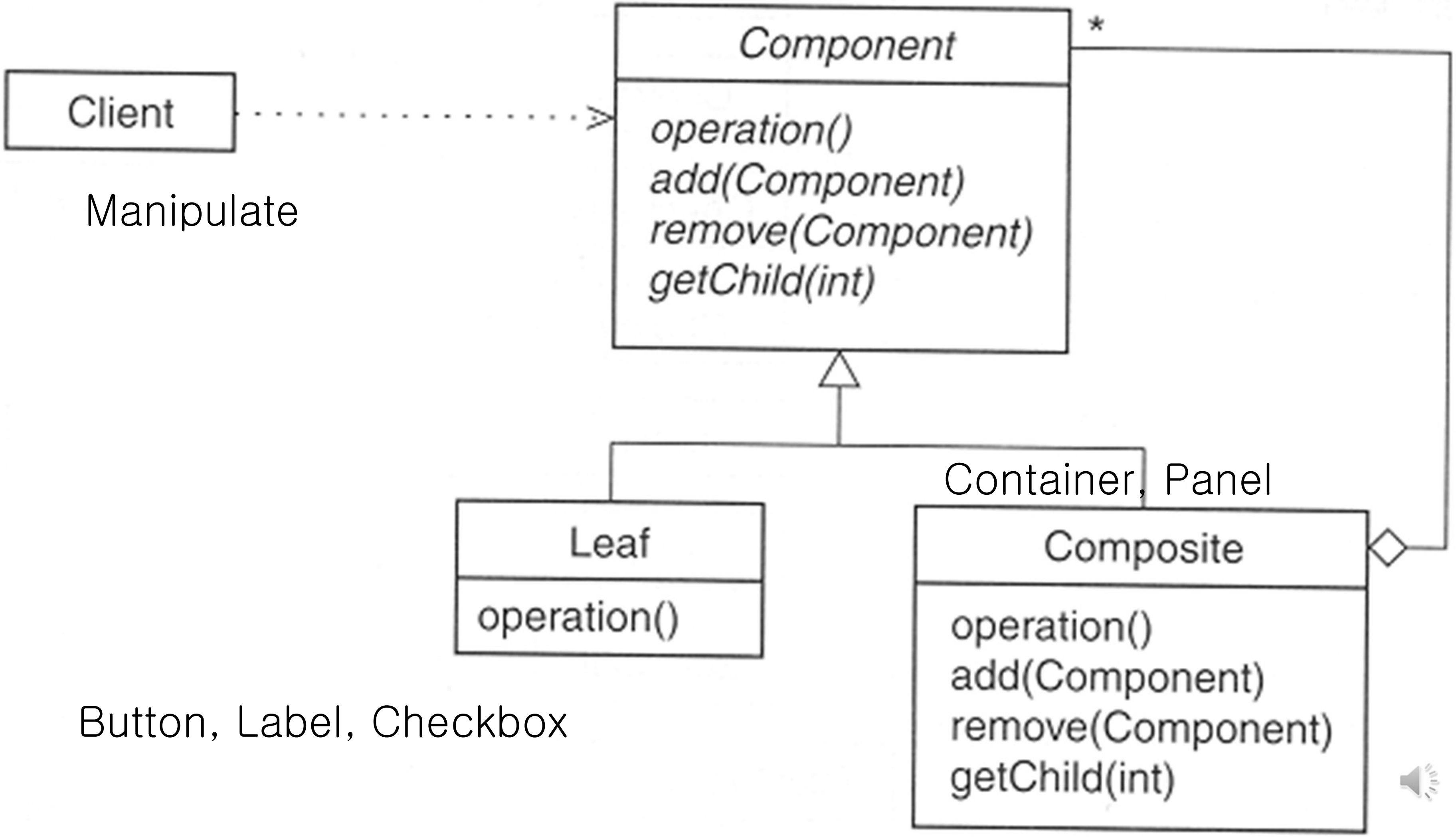
Category: Structural design pattern.

Intent: Compose objects into tree structures to represent a part or whole hierarchy. Composite lets clients treat individual objects and compositions of objects uniformly.

Applicability: Use the Composite pattern when

- you want to represent a part or whole hierarchy of objects; and
- you want clients to be able to ignore the difference between compositions of objects and individual objects (clients will treat all objects in the composite structure uniformly).

8.3.2 Design Pattern：Composite



Manipulate

Container, Panel

Button, Label, Checkbox

8.3.2 Design Pattern : Composite

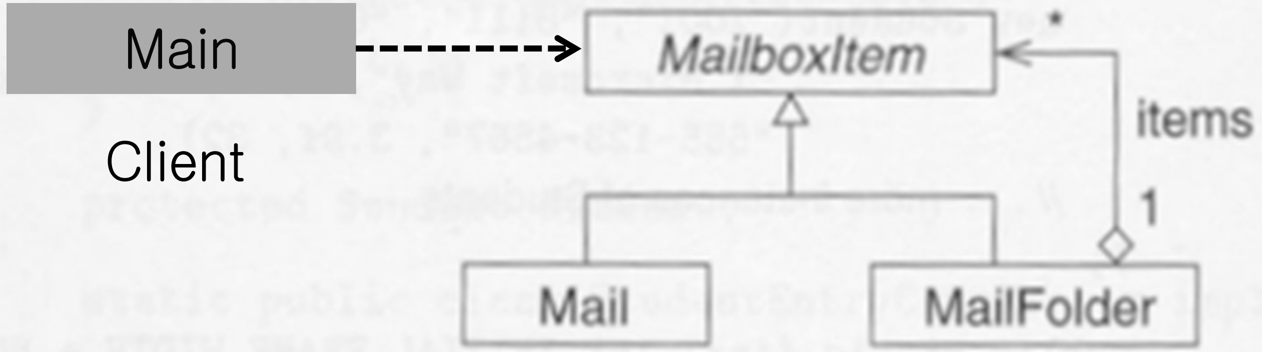
- The Participants of the Composite design pattern
  - Component
    - declares the interface for objects in the composition
    - implements default behavior for the interface
  - Leaf (e.g. Button, Label, CheckBox)
    - defines behavior for primitive objects in the composition
  - Composite(eg. Container and Panel)
    - declares an interface for accessing and managing its child components
    - defines behavior for components having children, stores child components
    - Implements child-related operations in the Component interface
  - Client
    - manipulates objects in the composition through the Component interface

10.4.2 Design Pattern : Composite (page 532)

- therefore, the mail folders form a hierarchy. The composite pattern can be used to represent such a hierarchical structure. The design of the e-mail manager program using the composite pattern is shown Figure 10.13

Figure 10.13

A design of the mail application using the Composite pattern.



MailBoxItem : the component role in Composite pattern

```
package mail;

public abstract class MailboxItem {

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public MailFolder getOwner() {
    return owner;
  }

  public void setOwner(MailFolder owner) {
    this.owner = owner;
  }

  public String toString() {
    return name;
  }

  public abstract int count(); //////////
  public abstract int countNewMail(); //////////

  protected MailboxItem(String name,
MailFolder owner) {
    this.name = name;
    this.owner = owner;
  }

  protected String name;
  protected MailFolder owner;

}
```

- Class mail.MailFolder (p. 532)
  - the MailFolder class is the composite role in Composite pattern

```
//* Composite(eg. Container and Panel)
 declares an interface for accessing and managing
its child components
defines behavior for components having children,
stores child components
Implements child-related operations in the
Component interface
*********************************************************/

package mail;

import java.util.*;

public class MailFolder extends MailboxItem {

  public MailFolder(String name) {
    super(name, null);
  }

  public MailFolder(String name, MailFolder owner) {
    super(name, owner);
  }

  public List getItems() {
    return items;
  }
```

```
public List getSubFolders() {
    List folders = new ArrayList();
    Iterator iterator = items.iterator();
    while (iterator.hasNext()) {
      Object item = iterator.next();
      if (item instanceof MailFolder) {
                folders.add(item);
      }
    }
    return folders;
  }

 public List getMails() {
    List mails = new ArrayList();
    Iterator iterator = items.iterator();
    while (iterator.hasNext()) {
      Object item = iterator.next();
      if (item instanceof Mail) {
                mails.add(item);
      }
    }
    return mails;
  }

  public void add(MailboxItem item) {
    items.add(item);
    item.setOwner(this);
  }

  public void add(MailboxItem[] items) {
    if (items != null) {
      for (int i = 0; i < items.length; i++) {
                add(items[i]);
      }
    }
  }
 public int count() {  //////////////
   int count = 0;
   Iterator iterator = items.iterator();
   while (iterator.hasNext()) {
     Object item = iterator.next();
     if (item instanceof MailboxItem) {
            count += ((MailboxItem) item).count();
     }
   }
   return count;
  }

  public int countNewMail() {          //////////////
   int count = 0;
   Iterator iterator = items.iterator();
   while (iterator.hasNext()) {
     Object item = iterator.next();
     if (item instanceof MailboxItem) {
            count += ((MailboxItem) item).countNewMail();
     }
   }
   return count;
  }

  protected List items = new ArrayList();

}
```

- Class mail.Mail (p. 534)
  - the Mail class is <u>the leaf role</u> in Composite pattern

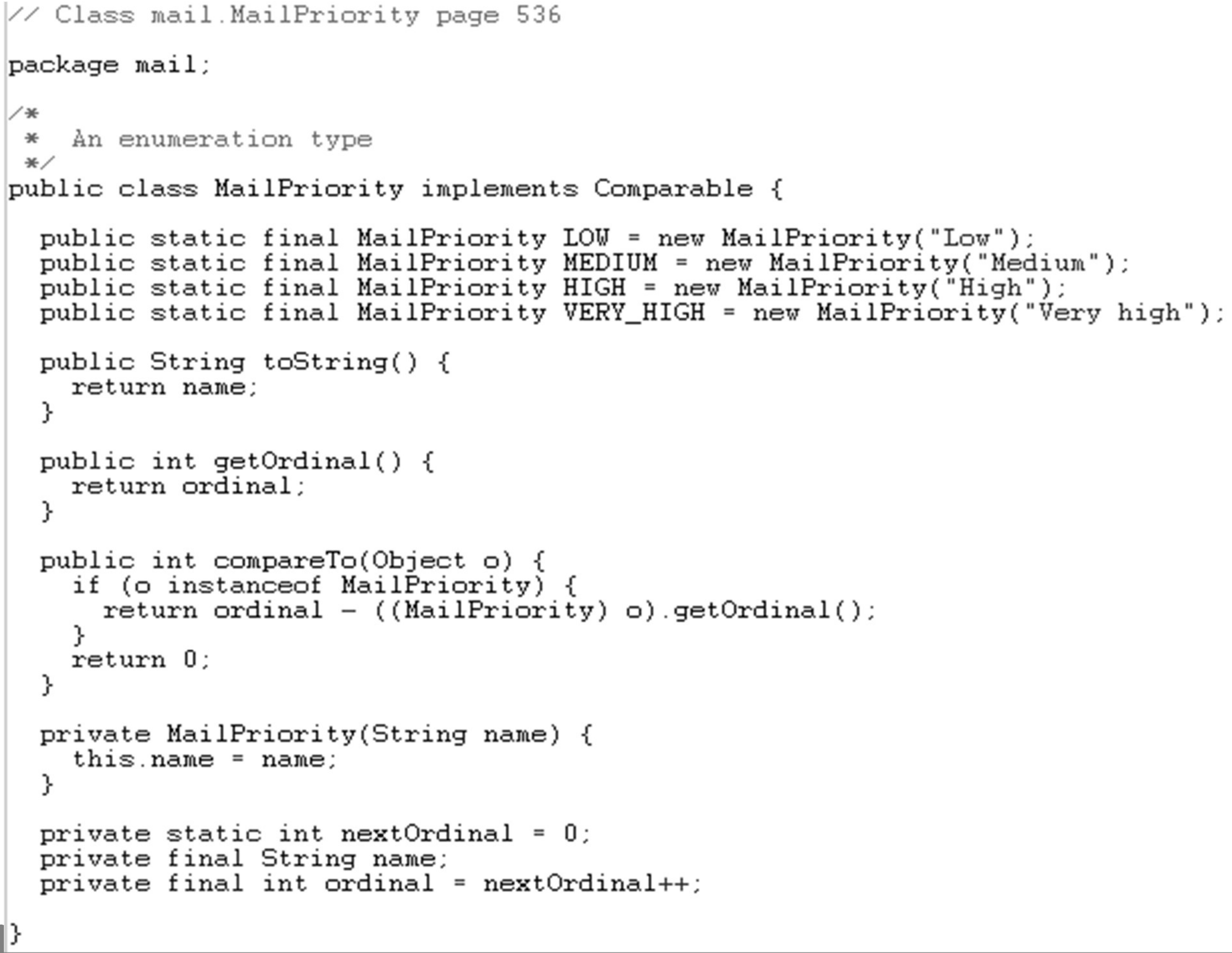
```
//* Leaf (e.g. Button, Label, CheckBox)
        defines behavior for primitive objects in
       the composition
***********************************/
package mail;
import java.util.*;

public class Mail extends MailboxItem {

  public Mail(String from,
                      String subject,
                      Date date,
                      MailPriority priority,
                      MailStatus status) {
    this(from, subject, null, date, priority, status, null,
null);
  }

  public Mail(String from,
                      String subject,
                      MailFolder owner,
                      Date date,
                      MailPriority priority,
                      MailStatus status,
                      String message,
                      List attachments) {
    super(subject, owner);
    this.from = from;
    this.date = date;
    this.priority = priority;
    this.status = status;
    this.message = message;
    this.attachments = attachments;
  }
```

```
public String getSubject() {
  return name;
 }

 public void setSubject(String subject) {
   name = subject;
 }

 public Date getDate() {
   return date;
 }

 public void setDate(Date date) {
   this.date = date;
 }

 public MailPriority getPriority() {
   return priority;
 }

 public void setPriority(MailPriority priority) {
   this.priority = priority;
 }

 public MailStatus getStatus() {
   return status;
 }

 public void setStatus(MailStatus status) {
   this.status = status;
 }
```

```
public String getFrom() {
  return from;
}

public void setFrom(String from) {
  this.from = from;
}

public String getMessage() {
  return message;
}

public void setMessage(String message) {
  this.message = message;
}

public List getAttachments() {
  return attachments;
}

public void addAttachment(Object attachment) {
  if (attachment != null) {
    attachments.add(attachment);
  }
}
```

```
public int count() {
                 ///////////
    return 1;
  }

  public int countNewMail() {                          ////////////
    return (status == MailStatus.NEW ? 1 : 0);
  }

  public String toString() {
    return "From: " + from + "; Subject: " + name +
";\n  Received: " + date +
    "; Priority: " + priority + "; Status: " + status +
";";
  }

  protected Date date;
  protected MailPriority priority;
  protected MailStatus status;
  protected String from;
  protected String message;
  protected List attachments;

}
```

- Mail.MailPriority (p. 536)
- Mail.MailStatus (p. 537)
  - two ordered enumeration types are used in the Mail class (MailPriority and MailStatus)



```
// Class mail.MailPriority page 536

package mail;

/*
 *  An enumeration type
 */
public class MailPriority implements Comparable {

  public static final MailPriority LOW = new MailPriority("Low");
  public static final MailPriority MEDIUM = new MailPriority("Medium");
  public static final MailPriority HIGH = new MailPriority("High");
  public static final MailPriority VERY_HIGH = new MailPriority("Very high");

  public String toString() {
    return name;
  }

  public int getOrdinal() {
    return ordinal;
  }

  public int compareTo(Object o) {
    if (o instanceof MailPriority) {
      return ordinal - ((MailPriority) o).getOrdinal();
    }
    return 0;
  }

  private MailPriority(String name) {
    this.name = name;
  }

  private static int nextOrdinal = 0;
  private final String name;
  private final int ordinal = nextOrdinal++;

}
```

```java
// Class mail.MailStatus page 537

package mail;

/*
 *  An enumeration type
 */
public class MailStatus implements Comparable {

  public static final MailStatus NEW = new MailStatus("New");
  public static final MailStatus READ = new MailStatus("Read");
  public static final MailStatus REPLIED = new MailStatus("Replied");
  public static final MailStatus FORWARDED = new MailStatus("Forwarded");

  public String toString() {
    return name;
  }

  public int getOrdinal() {
    return ordinal;
  }

  public int compareTo(Object o) {
    if (o instanceof MailStatus) {
      return ordinal - ((MailStatus) o).getOrdinal();
    }
    return 0;
  }

  private MailStatus(String name) {
    this.name = name;
  }

  private static int nextOrdinal = 0;
  private final String name;
  private final int ordinal = nextOrdinal++;

}
```

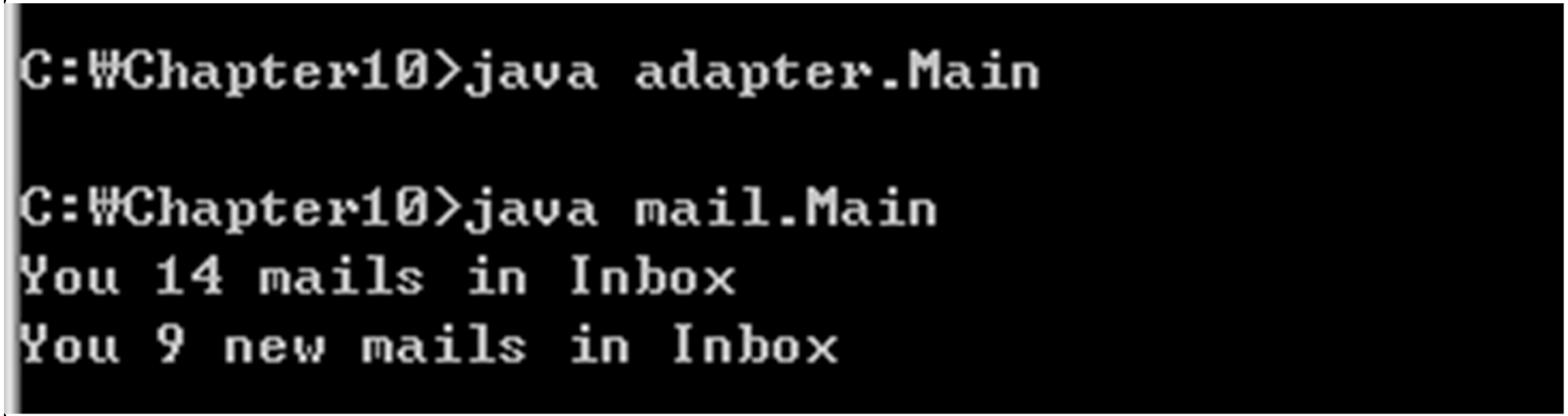
- The mail.Main Class
  - a simple test of the mail manager program
  - it populates the mailBox hierarchy by creating a number of a mail folders, subfolders, and mails
  - it gets a count of the total number of mails and the total number of new e-mail message in all folder, including nested subfolders, of the in box
  - mail.main (p. 537)

```
package mail;

import java.util.*;

public class Main {

  static Mail[] se450 = {
    new Mail("Bill Gates", "Need extension for final project",
                  getTime(2001, Calendar.NOVEMBER, 20, 23, 50),
                  MailPriority.VERY_HIGH, MailStatus.REPLIED),
    new Mail("Linus Torvalds", "Final project: Linux -- a small OS",
                  getTime(2001, Calendar.NOVEMBER, 19, 23, 39),
                  MailPriority.VERY_HIGH, MailStatus.REPLIED),
    new Mail("John Valissides", "Question on Facade pattern",
                  getTime(2001, Calendar.NOVEMBER, 18, 14, 56),
                  MailPriority.VERY_HIGH, MailStatus.REPLIED),
  };

  static Mail[] se452 = {
    new Mail("Marc Andreesen", "Problem with Netscape 7.02",
                  getTime(2001, Calendar.OCTOBER, 9, 23, 30),
                  MailPriority.HIGH, MailStatus.NEW),
    new Mail("James Gosling", "HW4 submission, version 3.14159",
                  getTime(2001, Calendar.OCTOBER, 2, 22, 40),
                  MailPriority.HIGH, MailStatus.NEW),
  };
```

```
static Mail[] work = {
  new Mail("Ralph Johnson", "CFP: PLOP",
             getTime(2001, Calendar.NOVEMBER, 16, 20, 10),
             MailPriority.HIGH, MailStatus.NEW),
  new Mail("Chris Galvin", "Your proposal",
             getTime(2001, Calendar.OCTOBER, 24, 4, 21),
             MailPriority.HIGH, MailStatus.NEW),
  new Mail("David Boise", "Abstract of my talk",
             getTime(2001, Calendar.DECEMBER, 2, 9, 24),
             MailPriority.HIGH, MailStatus.NEW),
 };

 static Mail[] news = {
  new Mail("WSJ.com", "Yahoo files for bankruptcy",
             getTime(2002, Calendar.JULY, 6, 7, 00),
             MailPriority.HIGH, MailStatus.NEW),
  new Mail("CERT", "Security alert, Windows XXP-3",
             getTime(2003, Calendar.JANUARY, 1, 0, 00),
             MailPriority.VERY_HIGH, MailStatus.NEW),
  new Mail("Scott McNealy", "Top 10 reasons for breaking up Microsoft",
             getTime(2001, Calendar.SEPTEMBER, 21, 9, 30),
             MailPriority.MEDIUM, MailStatus.READ),
  new Mail("Gordon Moore", "10 myths of Moors's law",
             getTime(2001, Calendar.SEPTEMBER, 11, 6, 23),
             MailPriority.MEDIUM, MailStatus.READ),
 };
```

```
static Mail[] junks = {
  new Mail("William Gates, III", ".NET Framework",
           getTime(2001, Calendar.OCTOBER, 26, 22, 12),
           MailPriority.HIGH, MailStatus.NEW),
  new Mail("AOL", "Free unlimited DSL with ...",
           getTime(2001, Calendar.OCTOBER, 2, 23, 59),
           MailPriority.HIGH, MailStatus.NEW),
};

public static MailFolder buildInbox() {
  MailFolder inboxFolder = new MailFolder("Inbox");
  MailFolder coursesFolder = new MailFolder("Courses");
  MailFolder se450Folder = new MailFolder("SE450");
  se450Folder.add(se450);
  MailFolder se452Folder = new MailFolder("SE452");
  se452Folder.add(se452);
  coursesFolder.add(se450Folder);
  coursesFolder.add(se452Folder);
  MailFolder workFolder = new MailFolder("Work");
  workFolder.add(work);
  MailFolder newsFolder = new MailFolder("News");
  newsFolder.add(news);
  MailFolder junksFolder = new MailFolder("Junk Mails");
  junksFolder.add(junks);
  inboxFolder.add(coursesFolder);
  inboxFolder.add(workFolder);
  inboxFolder.add(newsFolder);
  inboxFolder.add(junksFolder);
  return inboxFolder;
}
```

```
  public static void main(String[] args) {
    MailFolder inboxFolder = buildInbox();
    System.out.println("You " + inboxFolder.count() + " mails in Inbox");
    System.out.println("You " + inboxFolder.countNewMail() + " new mails in Inbox");
  }

  protected static Date getTime(int year, int month, int date, int hour, int minute) {
    Calendar calendar = Calendar.getInstance();
    calendar.set(year, month, date, hour, minute, 0);
    return calendar.getTime();
  }

}
```

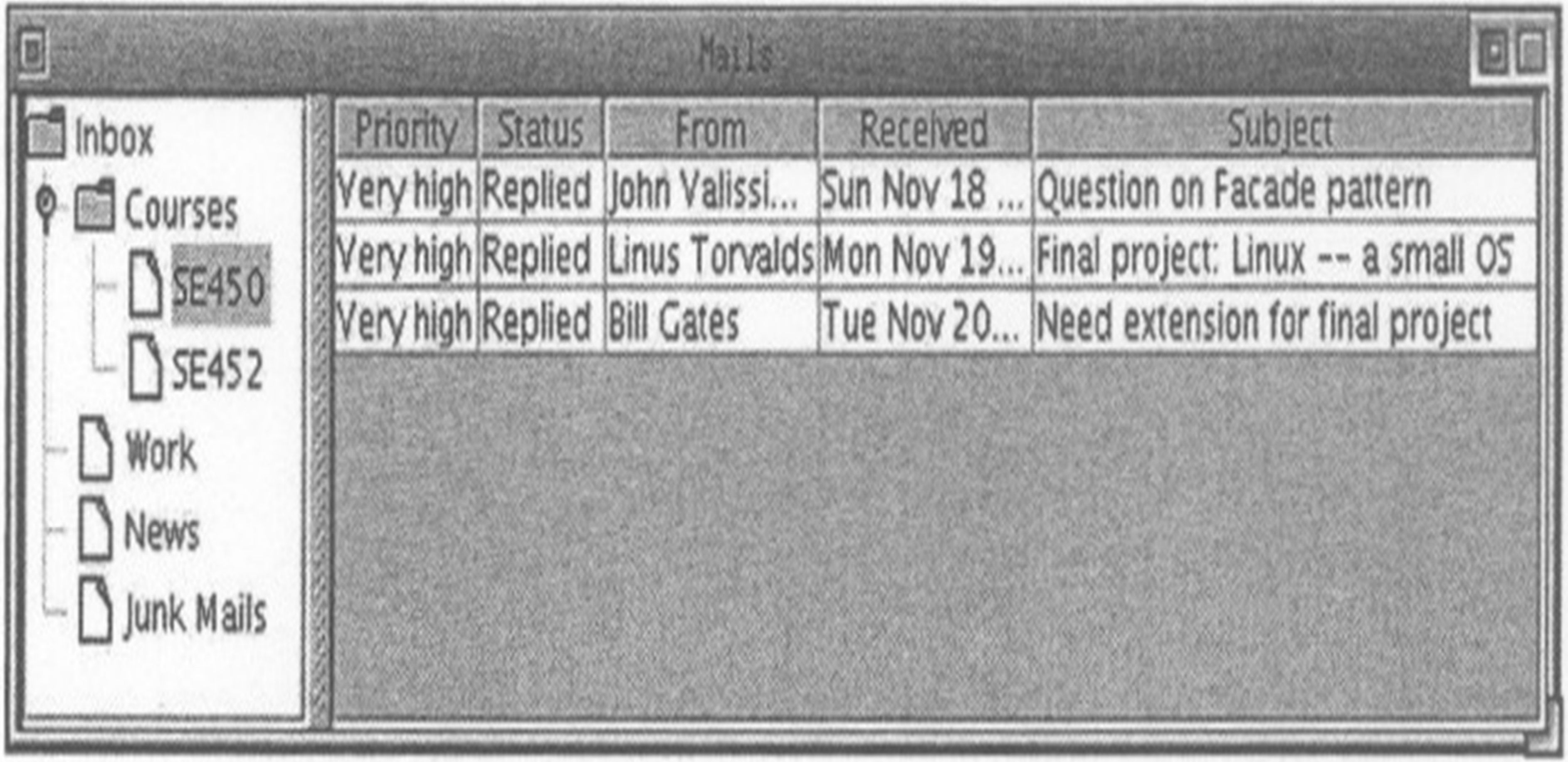


```
C:₩Chapter10>java adapter.Main

C:₩Chapter10>java mail.Main
You 14 mails in Inbox
You 9 new mails in Inbox
```

A GUI implementation

- The next step
  - to build a GUI interface.
  - the Hierarchy of folders is represented by a tree structure using the Jtree class in Swing
  - the mail in each folder is displayed using the generic table discussed in the previous section
  - A screen shot of the GUI interface in shown in Figure 10.14
  - The object Adapter pattern is used to adapt the Mail to TableEntry : Figure 10.15
    - the MailEntry is the adapter
    - the Mail class is the adaptee
    - the TableEntry interface is the target

Figure 10.14 page 539
A mail application



Priority	Status	From	Received	Subject
Very high	Replied	John Valissi...	Sun Nov 18 ...	Question on Facade pattern
Very high	Replied	Linus Torvalds	Mon Nov 19...	Final project: Linux -- a small OS
Very high	Replied	Bill Gates	Tue Nov 20...	Need extension for final project

Figure 10.15 page 539
A design of the GUI interface for the mail application using the Adapter pattern

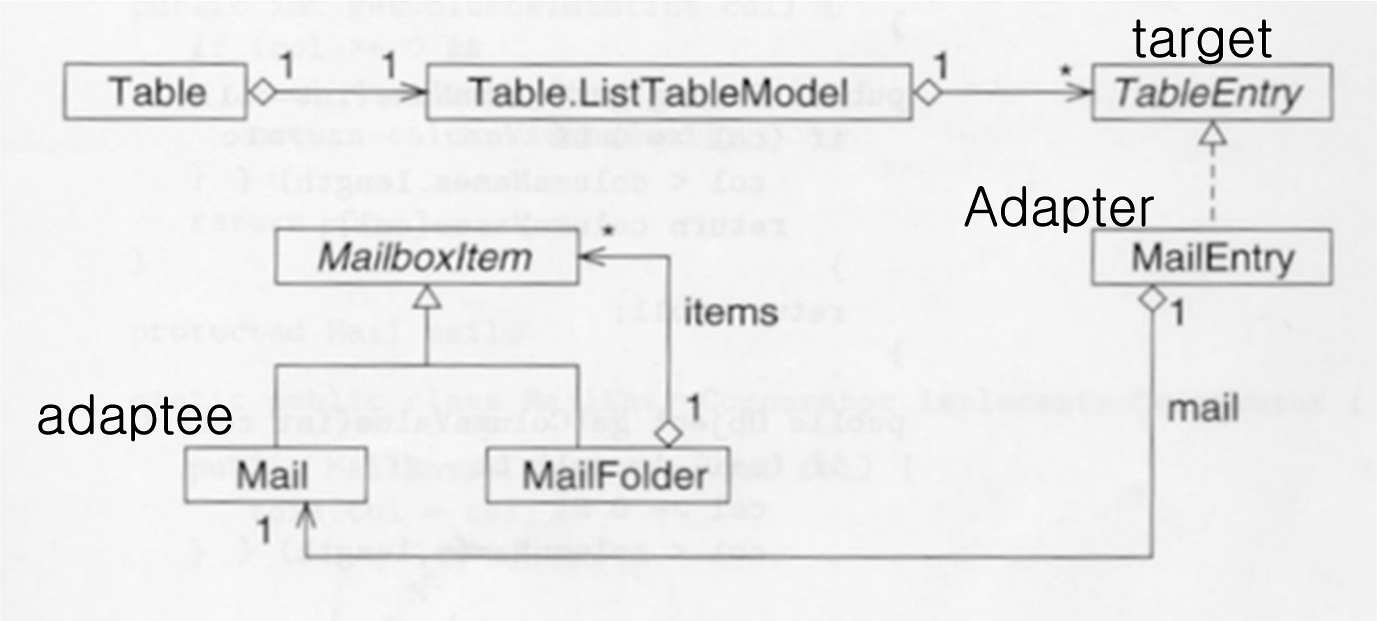
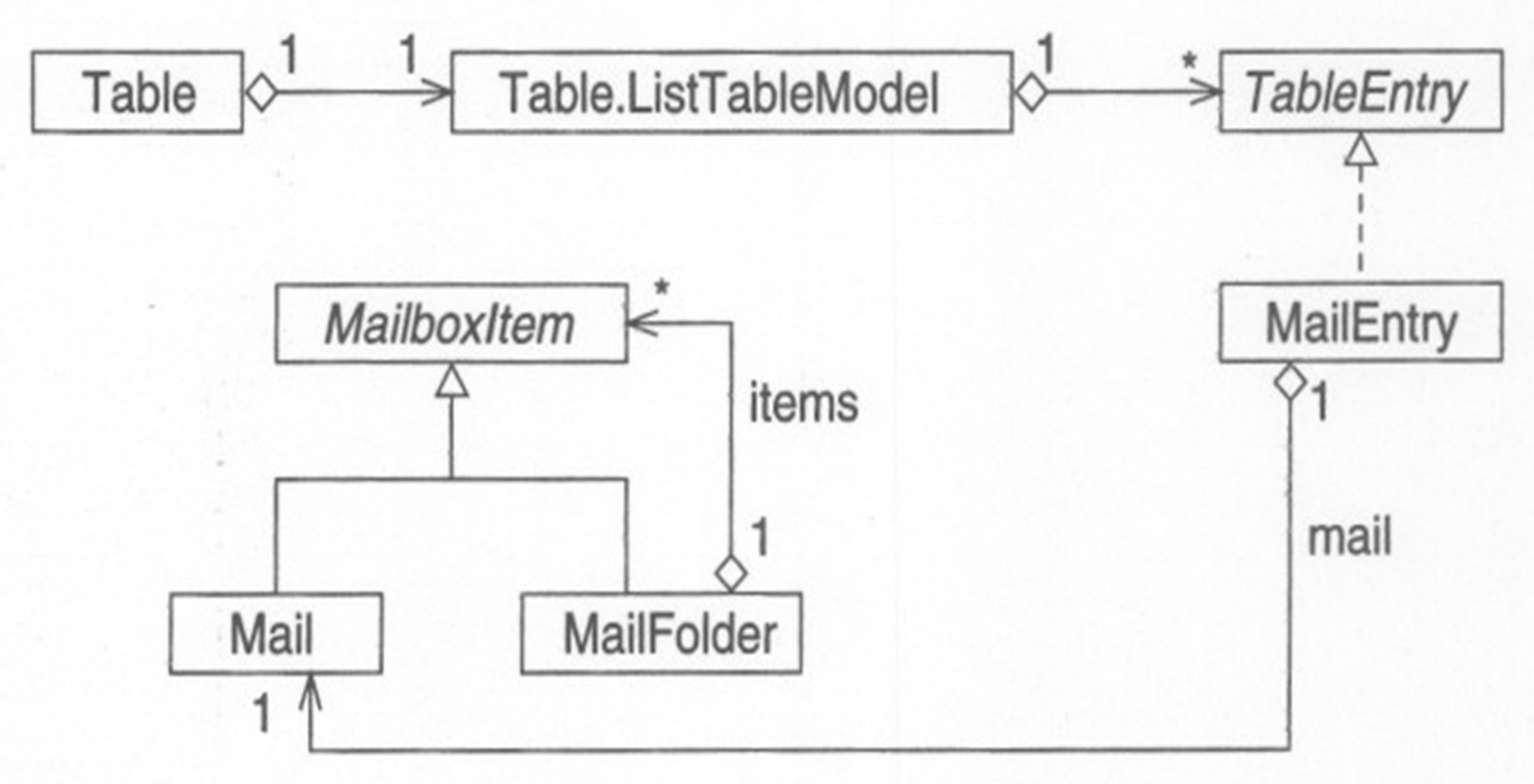


Figure 10.15 page 539
A design of the GUI interface for the mail application using the Adapter pattern



- Mail.gui.MailEntry (p. 539)
  -

```
package mail.gui;

import java.util.Date;
import java.util.Comparator;
import adapter.TableEntry;
import mail.*;

public class MailEntry implements TableEntry {

  // position of each column
  public static final int PRIORITY_COLUMN = 0;
  public static final int STATUS_COLUMN   = 1;
  public static final int FROM_COLUMN  = 2;
  public static final int RECEIVED_COLUMN = 3;
  public static final int SUBJECT_COLUMN  = 4;

  public static final String[] columnNames = {
    "Priority",
    "Status",
    "From",
    "Received",
    "Subject",
  };

  public static final String[] columnTips = {
    "Priority",
    "Status",
    "From",
    "Received",
    "Subject",
  };
```

```
public static final Comparator[] comparators = {
    new MailEntryComparator(PRIORITY_COLUMN),
    new MailEntryComparator(STATUS_COLUMN),
    new MailEntryComparator(FROM_COLUMN),
    new MailEntryComparator(RECEIVED_COLUMN),
    new MailEntryComparator(SUBJECT_COLUMN),
  };

  public static final int[] columnWidths = { 50, 50, 100, 150, 200};

  public MailEntry(Mail mail) {
    this.mail = mail;
  }

  public int getColumnCount() {
    return columnNames.length;
  }

  public String getColumnName(int col) {
    if (col >= 0 &&
                col < columnNames.length) {
      return columnNames[col];
    }
    return null;
  }
```

```
public Object getColumnValue(int col) {
   if (mail != null &&
            col >= 0 &&
            col < columnNames.length) {
     switch (col) {
     case PRIORITY_COLUMN: return mail.getPriority();
     case STATUS_COLUMN:   return mail.getStatus();
     case FROM_COLUMN:     return mail.getFrom();
     case RECEIVED_COLUMN: return mail.getDate();
     case SUBJECT_COLUMN:  return mail.getSubject();
     }
   }
   return null;
 }

 public String getColumnTip(int col) {
   if (col >= 0 &&
            col < columnTips.length) {
     return columnTips[col];
   }
   return null;
 }
```

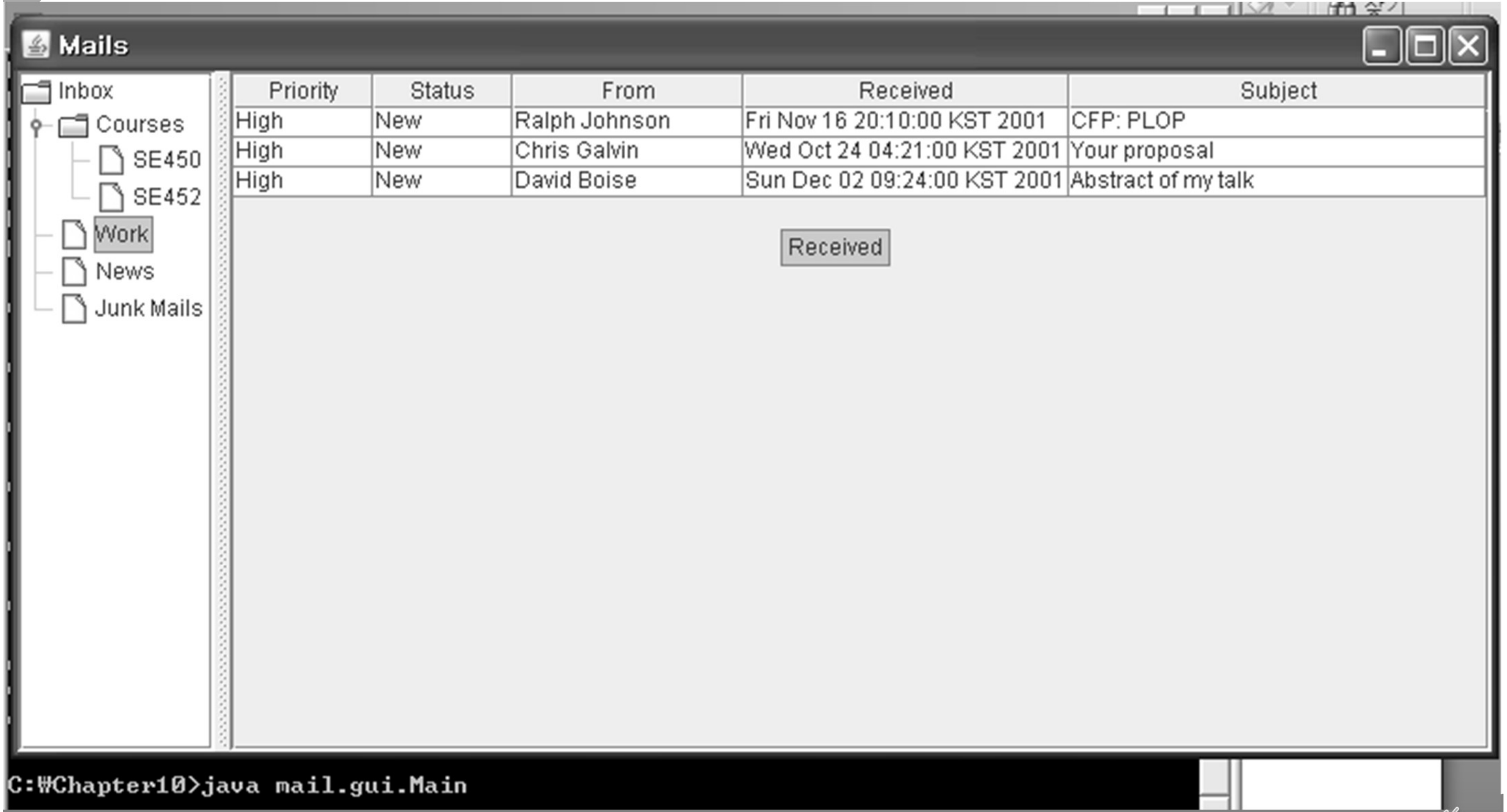
```
public Class getColumnClass(int col) {
  if (col == PRIORITY_COLUMN) {
   return MailPriority.class;
  } else if (col == STATUS_COLUMN) {
   return MailStatus.class;
  } else if (col == RECEIVED_COLUMN) {
   return Date.class;
  } else {
   return String.class;
  }
 }

 public Comparator getColumnComparator(int col) {
  if (col >= 0 &&
        col < comparators.length) {
   return comparators[col];
  }
  return null;
 }

 public int getColumnWidth(int col) {
  if (col >= 0 &&
        col < columnWidths.length) {
   return columnWidths[col];
  }
  return -1;
 }
```

```
protected Mail mail;

 static public class MailEntryComparator implements Comparator {

  public MailEntryComparator(int col) {
   this.col = col;
  }
  public int compare(Object o1, Object o2) {
   if (o1 != null &&
         o2 != null &&
         o1 instanceof MailEntry &&
         o2 instanceof MailEntry) {
        MailEntry e1 = (MailEntry) o1;
        MailEntry e2 = (MailEntry) o2;
        if (col == PRIORITY_COLUMN) {
         return -((Comparable)
e1.getColumnValue(col)).compareTo(e2.getColumnValue(col));
        } else if (col == STATUS_COLUMN) {
         return ((Comparable)
e1.getColumnValue(col)).compareTo(e2.getColumnValue(col));
        } else if (col == RECEIVED_COLUMN) {
         return ((Date) e1.getColumnValue(col)).compareTo(e2.getColumnValue(col));
        } else {
         return ((String) e1.getColumnValue(col)).compareTo(e2.getColumnValue(col));
        }
   }
   return 0;
  }
  protected int col;
 }
}
```

- Mail.gui.Main (p. 542)
  - constructs the GUI interface using Jtree Obect for the folder hierarchy and a Table object for displaying the mail in the current folder.

```
package mail.gui;

import java.awt.Dimension;
import java.awt.Toolkit;
import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;

import mail.*;
import adapter.*;

public class Main {

  public static final int INITIAL_FRAME_WIDTH = 800;
  public static final int INITIAL_FRAME_HEIGHT = 400;

  public static void main(String[] args) {
    MailFolder inboxFolder = mail.Main.buildInbox();
    JTree tree = new JTree(buildMailFolderTree(inboxFolder));
    JSplitPane splitPane = new JSplitPane();
    splitPane.setLeftComponent(new JScrollPane(tree));
    splitPane.setRightComponent(new JPanel());
    tree.addTreeSelectionListener(new MailFolderTreeSelectionListener(tree, splitPane));
JFrame frame = new JFrame("Mails");
    frame.setContentPane(splitPane);
    frame.setSize(INITIAL_FRAME_WIDTH, INITIAL_FRAME_HEIGHT);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.setLocation(screenSize.width / 2 - INITIAL_FRAME_WIDTH / 2,
                     screenSize.height / 2 - INITIAL_FRAME_HEIGHT / 2);
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  static class MailFolderTreeSelectionListener implements TreeSelectionListener {

    MailFolderTreeSelectionListener(JTree tree, JSplitPane splitPane) {
     this.tree = tree;
     this.splitPane = splitPane;
    }

    public void valueChanged(TreeSelectionEvent ev) {
     DefaultMutableTreeNode node =
          (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
     if (node != null) {
          Object item = node.getUserObject();
          if (item instanceof MailFolder) {
           splitPane.setRightComponent(new JScrollPane(buildTable((MailFolder) item)));
          }
     }
    }

    JTree tree;
    JSplitPane splitPane;
  }
```

```
protected static DefaultMutableTreeNode buildMailFolderTree(MailFolder folder) {
  if (folder != null) {
   DefaultMutableTreeNode root = new DefaultMutableTreeNode(folder);
   List subfolders = folder.getSubFolders();
   Iterator iterator = subfolders.iterator();
   while (iterator.hasNext()) {
        Object item = iterator.next();
        if (item instanceof MailFolder) {
         root.add(buildMailFolderTree((MailFolder) item));
        }
   }
   return root;
  }
  return null;
 }

 protected static Table buildTable(MailFolder folder) {
  if (folder != null) {
   List mails = folder.getMails();
   List entries = new ArrayList(mails.size());
   for (int i = 0; i < mails.size(); i++) {
        entries.add(new MailEntry((Mail) mails.get(i)));
   }
   return new Table(entries);
  }
  return null;
 }

}
```



Priority	Status	From	Received	Subject
High	New	Ralph Johnson	Fri Nov 16 20:10:00 KST 2001	CFP: PLOP
High	New	Chris Galvin	Wed Oct 24 04:21:00 KST 2001	Your proposal
High	New	David Boise	Sun Dec 02 09:24:00 KST 2001	Abstract of my talk

```
C:₩Chapter10>java mail.gui.Main
```

끝