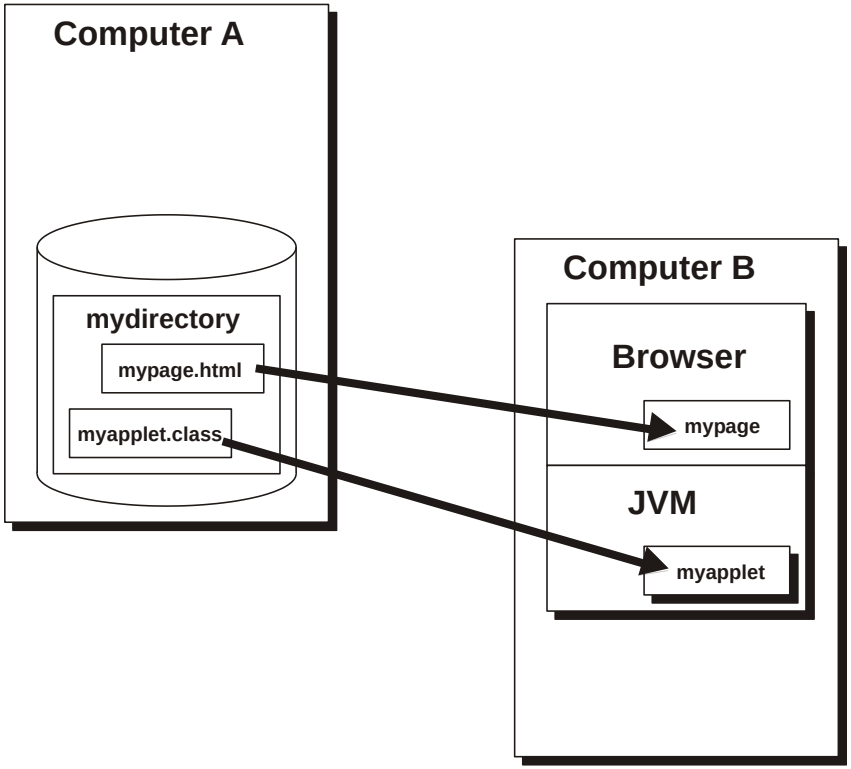
Network Computing Lecture 3

Professor Louise E. Moser

Spring 2012

More on Applets

- As we have seen in Lecture 2, an applet is a Java program that you can download from a server over the Internet and run on your local computer in
  - Web browser, or
  - Appletviewer
- Like any other object, an applet contains both
  - Code, and
  - Data



The Applet Lifecycle Methods

```
public class Simple extends JApplet {
        ...
        public void init() { . . . }
        public void start() { . . . }
        public void stop() { . . . }
        public void destroy() { . . . }
        ...
}
```

- The *init()* method contains the initialization code that, otherwise, would be put into the constructor of an object
- The *start()* method performs the applet's work or starts up one or more threads to do so
  - Most applets override the *start()* method provided by the *JApplet* class
- The *stop()* method is useful for stopping threads
  - If a user leaves the applet's page, it is appropriate to stop the animation threads
- The *destroy()* method allows an applet to release resources and close sockets cleanly

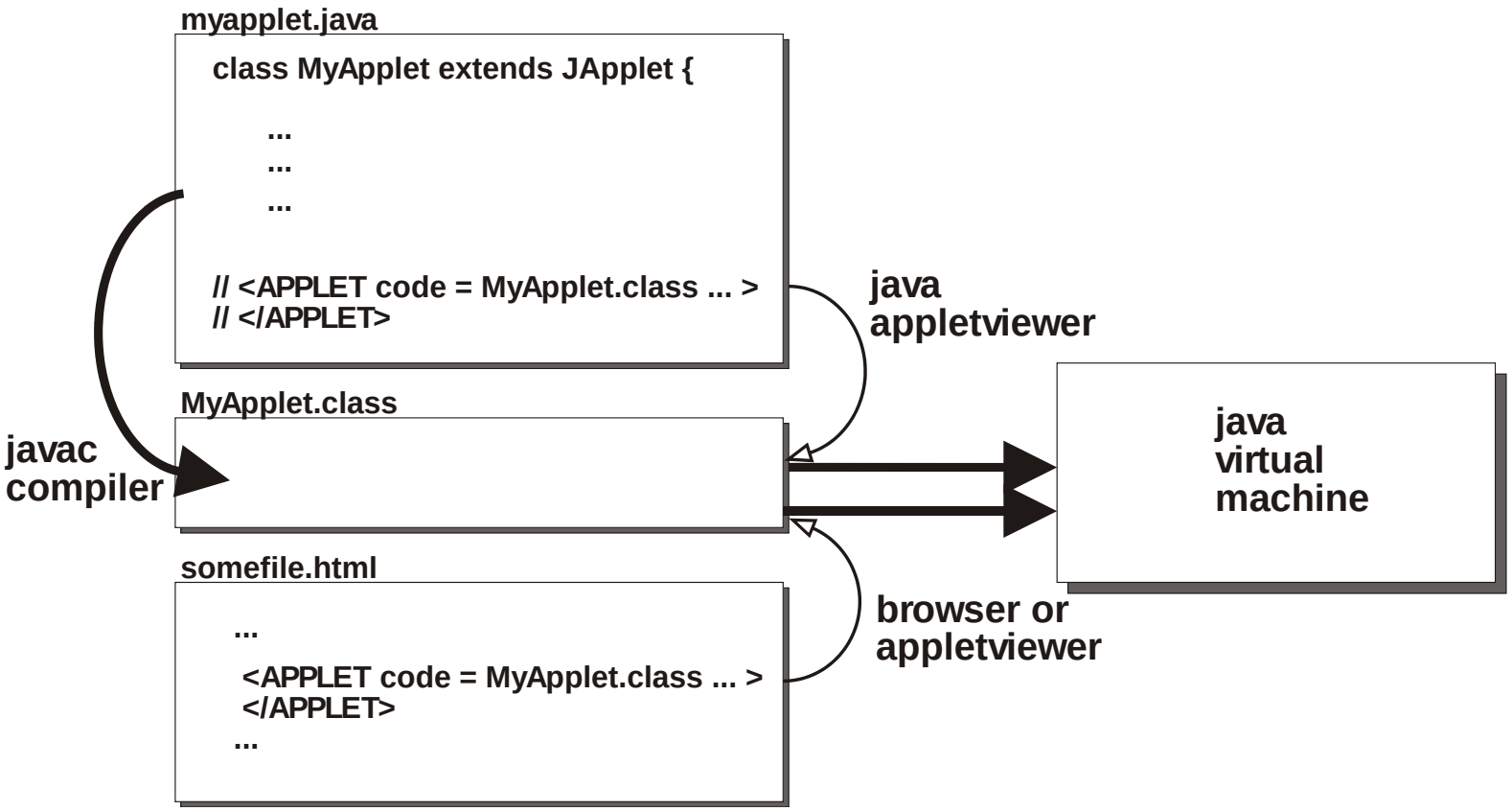
The Java Appletviewer

- The appletviewer is a program in the Java Development Kit (JDK) that you can use to test applets without a browser
- The appletviewer invokes applets from *.java* files or *.html* files
  - For a *.java* file that contains the Java code of one or more applets, the *appletviewer* searches for the <*APPLET*> tags and executes them, ignoring the Java code
  - For an *.html* file that contains <*APPLET*> tags, the *appletviewer* searches for the <*APPLET*> tags and executes them, ignoring other HTML commands
- In a *.java file*, <APPLET> tags are included as comments, e.g.,

  // <APPLET code=MyApplet.class width=200 height=100>

  // </APPLET>
- The *.java* file contains both the Java code for the applets and also the commented <*APPLET*> tags that are used for testing

The Java Appletviewer



The Rolodex Example

- A **Rolodex** is a database that contains the names of people and information about them such as their birthdays, phone numbers, email addresses, snail mail addresses, etc
- The information is indexed by the **person's name**
- You enter the person's name and the Rolodex displays the available information about the person
- If it does not contain an entry for the person, the Rolodex displays blank information
- The data field can be edited by the user to include additional information about the person, which is then recorded in the database

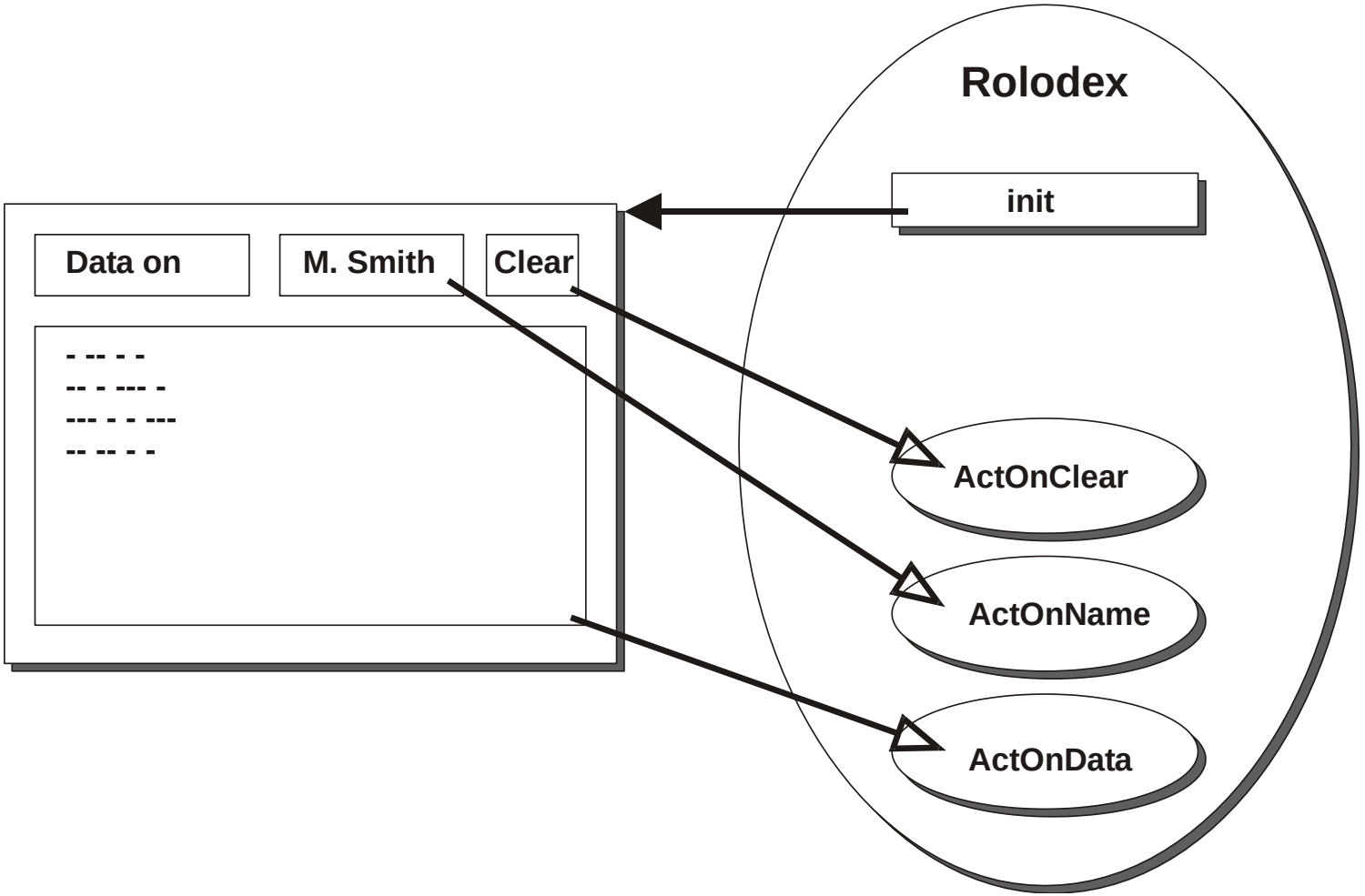
Handling Events in the Rolodex

- We need to handle events when the user clicks a *JButton* or edits a *JTextField* or a *JTextArea*
- An easy way to handle events in Java is to use **inner classes**, i.e., classes that are declared within a class
- When we create a new *JButton*, a new *JTextField*, etc, we
  - Create a new inner class to handle events for the new object, and
  - Attach an instance of that class to act as an **event listener** for the object

Handling Events in the Rolodex

- In the Rolodex example, we have four objects in the GUI
  - Three of these objects can be modified or clicked by the user
    - *clearButton JButton*
    - *nameText JTextField*
    - *nameData JTextArea*
  - Thus, there are three inner classes to handle three kinds of events
    - *ActOnClear*
    - *ActOnName*
    - *ActOnData*

The Rolodex Example



The Rolodex Applet

```
public class Rolodex extends JApplet {
    ...
    private JLabel rolodexLabel = new JLabel(waitingMessage);
    private JButton clearButton = new JButton("Clear");
    private JTextField nameText = new JTextField("");
    private JTextArea nameData = new JTextArea("", 10, 30);
```

- In this example, we have
  - *JLabel* for a message
  - *JButton* for clearing
  - *JTextField* for a name
  - *JTextArea* for the data

The ActOnClear Inner Class

```
// Invoked when the user clicks on the Clear button
// to display the label "waiting for a name"  and
// to erase the text and data fields
class ActOnClear implements ActionListener {
  public void actionPerformed(ActionEvent e) {
      rolodexLabel = new JLabel(waitingMessage);
      nameText = new JTextField("");
      nameData = new JTextArea("", 10, 30);
  }
}
```

- *init()* attaches the event listener *ActOnClear* to the event source *clearButton* as follows:

```
clearButton.addActionListener(new ActOnClear());
```

The ActOnName Inner Class

```
// Invoked when the user enters a name in nameText
// in the JTextField to display the label "name is" and
// to get the name using the getText() method and then
// to find the data associated with that name and display it
class ActOnName implements ActionListener {
  public void actionPerformed(ActionEvent e) {
      rolodexLabel = new JLabel(dataMessage);
      requestedName = nameText.getText();
      // Find the data, in the rolodex database, corresponding to
      // the name and display the data
  }
}
```

- *init()* attaches the event listener *ActOnName* to the event source *nameText* as follows:

```
nameText.addActionListener(new ActOnName());
```

The ActOnData Inner Class

```
// Invoked when the user edits the data in nameData
// in the JTextArea to record the edited data in the database
class ActOnData implements TextListener {
  public void textValueChanged(TextEvent e) {
      // Extract data from nameData and record the data
      // in the database, corresponding to the name
      // in nameText
      }
}
```

- *init()* attaches the event listener *ActOnData* to the event source *nameData* as follows:

  ```
  nameData.addTextListener(new ActOnData());
  ```

The init() Method

- *init()* creates a BorderLayout GUI and then attaches the **event listeners** to the **event sources**

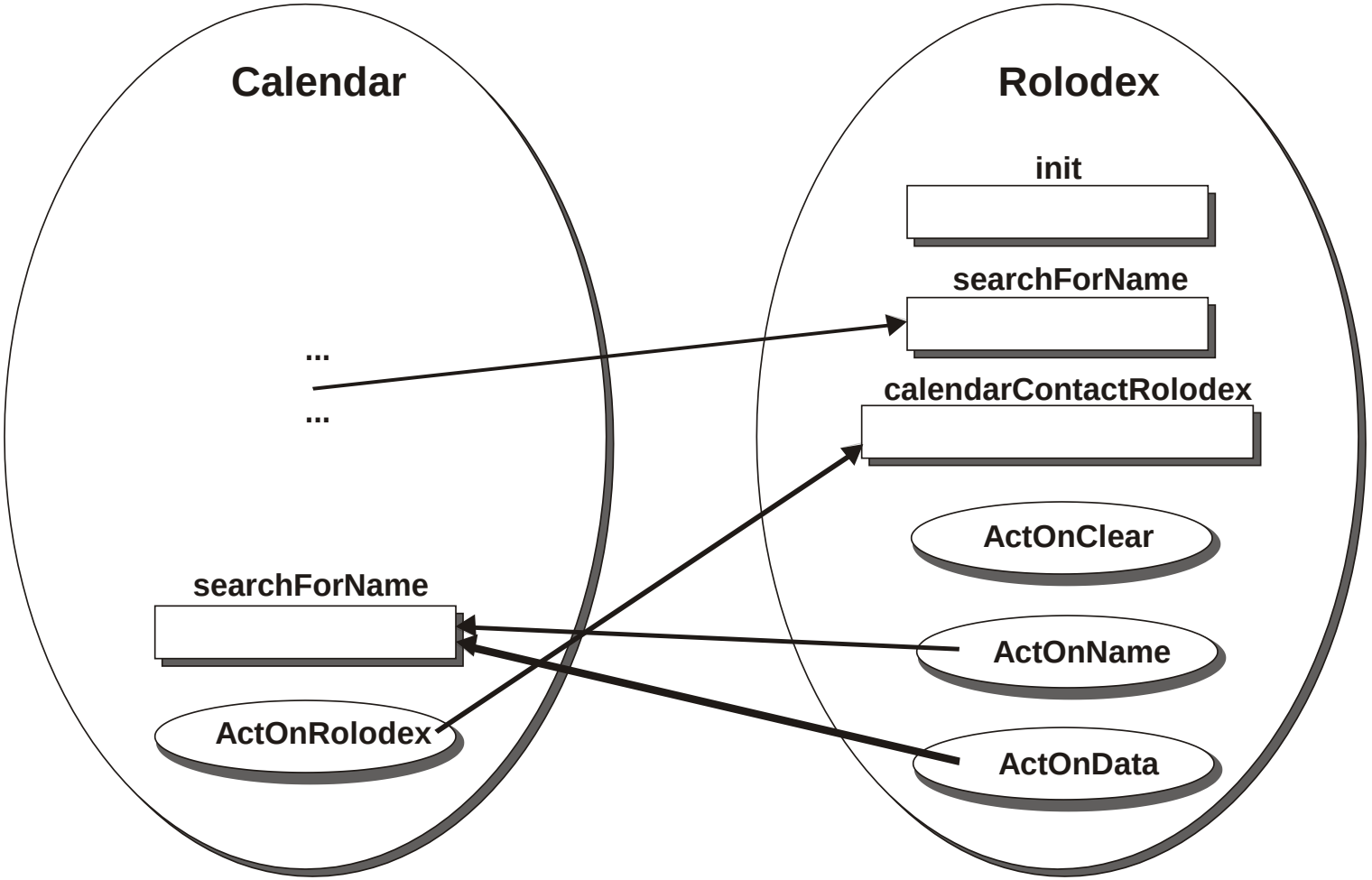
```
public void init() {
    setLayout(new BorderLayout());
    topPanel.add(rolodexLabel);
    topPanel.add(nameText);
    topPanel.add(clearButton);
    add("North", topPanel);

    bottomPanel.add(nameData);
    add("South", bottomPanel);

    clearButton.addActionListener(new ActOnClear());
    nameText.addActionListener(new ActOnName());
    nameData.addTextListener(new ActOnData());
}
```

- Note that there is no *start()* method
- All actions are initiated by the user through **events**

Connecting the Rolodex to the Calendar

- As we discussed in Lecture 2, you can connect two applets together, provided that
  - The applets originate from the same directory on the same server (the same codebase), and
  - The applets are running on the same Web page in the same browser window
- We have a *rolodex* applet that can be used free-standing
- We also have a *calendar* applet that can be used free-standing
- We can cause the *rolodex* applet to connect to the *calendar* applet
- If we select the name of a person in the *rolodex*, the *calendar* searches for the next appointment with that person
- If we create or click on an appointment in the *calendar*, the *rolodex* searches for information about the person

The Big Connection



Augmenting the Declarations of the Rolodex Class

```
public class Rolodex extends JApplet {

    ....

    private JApplet calendar = null;

    private Boolean calendarLinked = false;
```

- To connect the *rolodex* applet to the *calendar* applet we augment the declarations of the *Rolodex* class to include
  - A *calendar* applet reference
  - A boolean to indicate that the *rolodex* is linked to the *calendar*
- The *calendar* applet has similar declarations

Making the Connection

- Next, we cause the *rolodex* applet to connect to the *calendar* applet
- To do so, the *rolodex* applet needs to know the address of the *calendar* applet
- The *calendar* applet has a button named *rolodexButton*
- When the user clicks this button, the *calendar* applet sends its address to the *rolodex* applet
- Note that the *calendar* applet knows the name "Rolodex" and gets the address of the *rolodex* applet by invoking *getAppletContext("Rolodex")*

Making the Connection

- Thus, the *calendar* applet invokes the *calendarContactRolodex* method of the *rolodex* applet and includes a reference *a* to itself as a parameter

```
// Calendar applet initiates cooperation with Rolodex applet
public void calendarContactRolodex(JApplet a) {
    if (a instanceof Calendar) {
      calendar = a;
      calendarLinked = true;
} }
```

- The *instanceof* operator checks that the given applet is indeed a C*alendar* applet
- Note that *instanceof* is not *instanceOf*

Invoking the Calendar within ActOnName

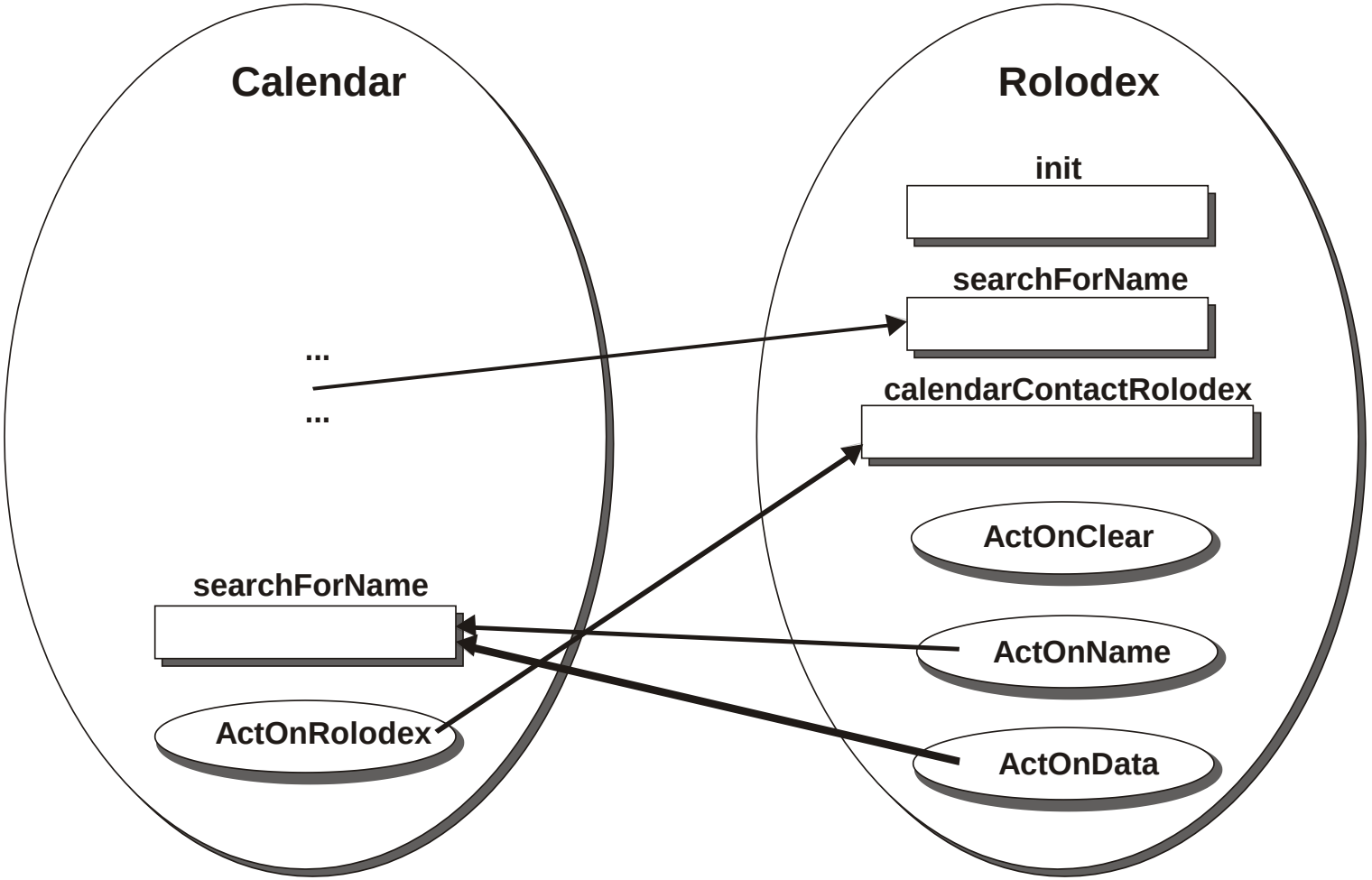
```
class ActOnName implements ActionListener {
  public void actionPerformed(ActionEvent e) {
     rolodexLabel = new JLabel(dataMessage);
     requestedName = nameText.getText();
     // Find the data in the rolodex database
     // and display the data
     if (calendarLinked) {
       try { // Invoke the calendar to search for the next appointment with the person
       (Calendar)calendar.searchForName(requestedName)
       }
       catch (IllegalAccessException) {
         // Calendar has been closed, nullify reference
         calendar = null;
         calendarLinked = false;
    } }
} }
```

- Note the test of *calendarLinked* and also the catching of the exception if *calendar* has been closed and thus cannot be invoked

Connecting the Calendar to the Rolodex

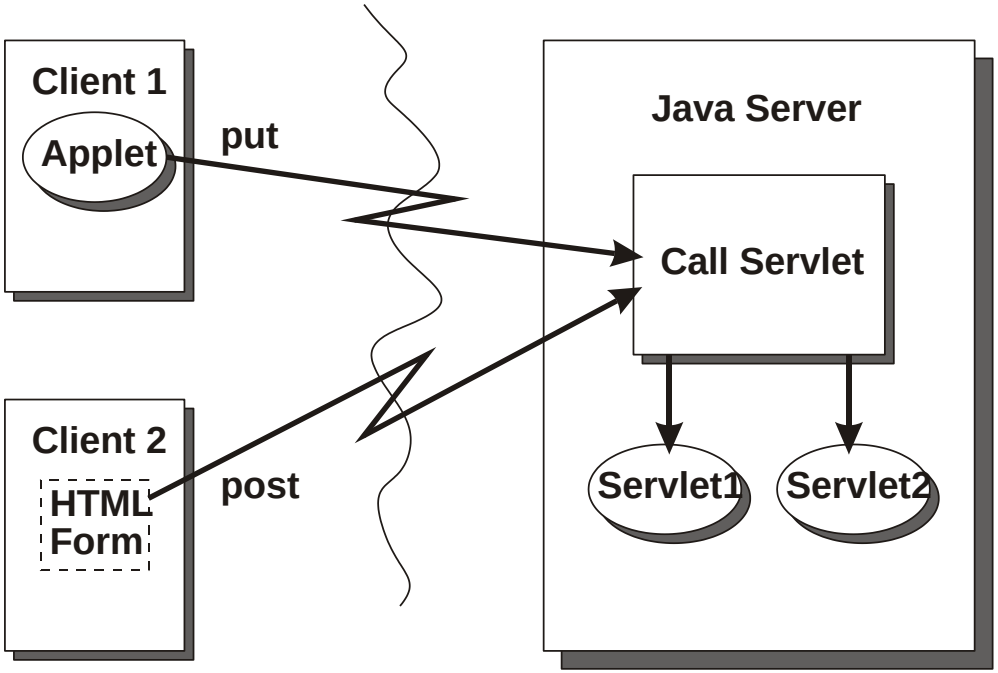
- On the previous slide, when we select the name of a person in the rolodex, the *calendar* applet is invoked to search for the next appointment with that person
- Similarly, when we click on an appointment in the calendar, the *rolodex* applet is invoked to search for information about the person with that appointment
  - The *calendar* applet has a button named *rolodexButton*
    - When the user clicks this button, the *calendar* applet looks for an applet named "ROLODEX" and then connects to it
    - The inner class *ActOnRolodex* handles the button click
  - The *calendar* applet causes the *rolodex* applet to search for information in the rolodex about the person with the appointment by invoking
    - *rolodex.searchForName()*

The Big Connection



Applets vs. Servlets

- Applets run on a client machine
- Servlets run on a server machine



Servlets

- Servlets are the server-side equivalent of applets; servlets are used to create dynamic Web pages
- A servlet can be loaded dynamically onto a server, where it responds to requests from applets or other Java applications
- A servlet usually originates on the server, rather than being uploaded from the client
  - However, security restrictions do apply to uploaded servlets
- Servlets can do lots of useful things that applets cannot do, including
  - Reading and writing files
  - Talking to applets on multiple computers
- We will learn more about servlets later