About This Lecture

- In this lecture we will learn about an Interface called List and write an example program that uses one of its implementations.

Outline

- Lists
- Structure Interface Hierarchy for List
- Util Interface Hierarchy for List
- The Store, Collection and List Interfaces
- Example of Using Lists

A List

- A List is a non-indexed container object that can grow and shrink at either end.

Structure Collection Hierarchy

- In the structure package and in java.util, Vectors and Arrays are containers, but they are not called collections.
- The structure package defines the following interface hierarchy.

Util Collection Hierarchy

- In the java.util package, there is no Store interface.
- In addition, the interfaces Collection and List have more messages defined than in the structure package.
- The java.util package defines the following interface hierarchy.
Structure Interface - Store

```java
public interface Store {
    public int size();
    // post: returns the number of elements contained in
    // the store.
    public boolean isEmpty();
    // post: returns the true iff store is empty.
    public void clear();
    // post: clears the store so that it contains no
    // elements.
}
```

code based on Bailey pg. 18

Structure Interface - Collection

```java
public interface Collection extends Store {
    public boolean contains(Object anObject);
    // pre: anObject is non-null
    // post: returns true iff the collection contains the object
    public void add(Object anObject);
    // pre: anObject is non-null
    // post: the object is added to the collection. The
    // replacement policy is not specified
    public Object remove(Object anObject);
    // pre: anObject is non-null
    // post: removes object "equal" to anObject and returns it,
    // otherwise returns null
    public Iterator elements(); // Ignore until iterator lecture
    // post: return an iterator for traversing the collection
}
```

code based on Bailey pg. 19

Structure Interface - List 1

```java
public interface List extends Collection {
    public void addToHead(Object anObject);
    // pre: anObject is non-null
    // post: the object is added to the beginning of the list
    public void addToTail(Object anObject);
    // pre: anObject is non-null
    // post: the object is added to the end of the list
    // Note that the replacement policy for the method
    // add(Object) is specified in List as: addToTail(Object)
    public Object peek();
    // pre: list is not empty
    // post: returns the first object in the list without
    // modifying the list
}
```

code based on Bailey pg. 99

Structure Interface - List 2

```java
public Object tailPeek();
    // pre: list is not empty
    // post: returns the last object in the list without
    // modifying the list
    public Object removeFromHead();
    // pre: list is not empty
    // post: removes and returns first object from the list
    public Object removeFromTail();
    // pre: list is not empty
    // post: removes and returns last object from the list
}
```

code based on Bailey pg. 99

List Example - E-mail Handler

- Consider an application that gathers e-mail from multiple clients and sends the messages to external destinations, one at a time.
- We will define an interface for this application and implement the interface using a class that uses a List.
- We will also build a simple test harness for this application to make sure that it works.

E-mail Handler - Capabilities

- Add an urgent e-mail
- Add a normal e-mail
- Send an e-mail
- Check if an e-mail has not been sent yet
- Abort an e-mail that is waiting to be sent
E-mail Handler Interface -1

```java
public interface EmailHandler {
    public void addNormal(Email anEmail);
    // post: Add the given Email as an Email that is waiting to
    // be sent. This Email must be sent after all of the other
    // Emails that are already waiting.

    public void addUrgent(Email anEmail);
    // post: Add the given Email as an Email that is waiting to
    // be sent. This Email must be sent before all of the other
    // Emails that were added using the addNormal(Email) method
    // and all of the other Emails that are subsequently added
    // using the addNormal(Email) method.
}
```

E-mail Handler Interface -2

```java
public void send();
// post: If there is at least one Email waiting to be sent
// then send one Email and return an object that represents
// it. If none are waiting, return null. The Email that is
// sent must be any urgent Email if an urgent Email is
// waiting. Otherwise, it must be the normal Email that has
// been waiting the longest.

public void abort(String sender, String destination);
// post: If an Email with the given sender and destination
// is waiting to be sent, then return an object that
// represents it and never send it. If no such Email is
// waiting, return null. If more than one such Email is
// waiting, abort any one of them.
```

E-mail Handler Interface - 3

```java
public boolean isWaiting(String sender, String destination);
// post: If an Email with the given sender and destination
// is waiting to be sent, then return true. If no such Email
// is waiting, return false.
```

A List Implementation of the E-mail Handler

- We can use a single List to implement the E-mail Handler.

  Add a normal e-mail using `addToTail(object)`
  Add an urgent e-mail using `addToHead(Object)`
  Send an e-mail using `removeFromHead()`
  Check if an e-mail is waiting to be sent using `contains(Object)`
  Abort an e-mail that has not been sent yet using `remove(Object)`

EmailHandlerList - Constructor

```java
public class EmailHandlerList implements EmailHandler {
    private List list;
    public EmailHandlerList() {
        // post: Initialize the new EmailHandlerList to have no
        // waiting Emails.
        this.list = new SinglyLinkedList();
        // SinglyLinkedList is one Class that implements the
        // List interface.
    }
}
```

EmailHandlerList - add

```java
public void addNormal(Email anEmail) {
    // post: Add the given Email as an Email that is waiting to
    // be sent. This Email must be sent after all of the other
    // Emails that are already waiting.
    this.list.addToTail(anEmail);
}

public void addUrgent(Email anEmail) {
    // post: Add the given Email as an Email that is waiting to
    // be sent. This Email must be sent before all of the other
    // Emails that were added using the addNormal(Email) method
    // and all of the other Emails that are subsequently added
    // using the addNormal(Email) method.
    this.list.addToHead(anEmail);
}
```
EmailHandlerList Class - send

```java
public Email send() {
   // post: If there is at least one Email waiting to be sent
   // then send one Email and return an object that represents
   // it. If none are waiting, return null. The Email that is
   // sent must be any urgent Email if an urgent Email is
   // waiting. Otherwise, it must be the normal Email that has
   // been waiting the longest.
   if (this.list.isEmpty())
      return null;
   else
      return (Email) this.list.removeFromHead();
}
```

EmailHandlerList Class - isWaiting

```java
public boolean isWaiting(String sender, String destination) {
   // post: If an Email with the given sender and destination
   // is waiting to be sent, then return true. If no such Email
   // is waiting, return false.
   Email anEmail;
   anEmail = new Email(sender, destination, "");
   // We are relying on an implementation of
   // equals(Object) in the Email class that returns true
   // if two Email objects have equal sender Strings and
   // equal destination Strings.
   return this.list.contains(anEmail);
}
```

EmailHandlerList Class - abort

```java
public Email abort(String sender, String destination) {
   // post: If an Email with the given sender and destination
   // is waiting to be sent, then return an object that
   // represents it and never send it. If no such Email is
   // waiting, return null. If more than one such Email is
   // waiting, abort any one of them.
   Email anEmail;
   anEmail = new Email(sender, destination, "");
   // We are relying on an implementation of
   // equals(Object) in the Email class that returns true
   // if two Email objects have equal sender Strings and
   // equal destination Strings.
   return (Email) this.list.remove(anEmail);
}
```

EmailHandlerList Class - toString

```java
public String toString() {
   // post: Return a String representation of the receiver.
   // This method is included so we can output the Emails
   // that are waiting.
   return this.list.toString();
}
```

An Implementation of the Email Class

- We can use an object with three fields to implement an Email.
- In fact, we can add as many other fields as we want, like a date and time, etc.

```java
public class Email {
   private String sender;
   private String destination;
   private String body;
   public Email(String from, String to, String message) {
      // post: Initialize the new Email to have the given
      // sender, destination and body Strings.
      this.sender = from;
      this.destination = to;
      this.body = message;
   }
}
```
Email - equals

```java
public boolean equals(Object another) {
    // post: return true iff the given Object is equivalent
to the receiver object
    Email anEmail;
    if (! (another instanceof Email))
        return false;
    anEmail = (Email) another;
    return this.sender.equals(anEmail.sender) &&
            this.destination.equals(anEmail.destination);
}
```

Email - toString

```java
public String toString() {
    // post: return a String representation of the receiver
    return "From: " + this.sender + "\n" +
           "To: " + this.destination + "\n" +
           this.body + "\n";
}
```

A Test Harness for E-mail Handler

- We can write a command line driver program to
test the E-mail Handler.
- We can take input either from a file or from the
  keyboard.
- We will use the keyboard.
- We will send a series of commands to the E-mail
  handler, where each command triggers one of the
  5 methods: addNormal(Email), addUrgent(Email),
send(), abort(String, String) or isWaiting(String,
String).
- After each command we will output the Emails
  that are waiting.

TestEmailHandler - main 1

```java
public class TestEmailHandler {
    public static void main(String args[]) {
        String command;
        String sender;
        String dest;
        EmailHandler handler;
        handler = new EmailHandlerList();
        sender = "";
        dest = "";
        System.out.println(handler);
        while (true) {
            // infinite loop
            command = TestEmailHandler.getString("Command");
            if (command.equals("stop")) {
                System.out.println("End Test.");
                return;
            }...
```

TestEmailHandler - main 2

```java
else if (command.equals("send"))
    TestEmailHandler.send(handler);
else if (command.equals("abort"))
    TestEmailHandler.abort(handler, sender, dest);
else if (command.equals("check"))
    TestEmailHandler.check(handler, sender, dest);
else if (command.equals("normal"))
    TestEmailHandler.normal(handler, sender, dest);
else if (command.equals("urgent"))
    TestEmailHandler.urgent(handler, sender, dest);
else
    System.out.println("Invalid Command. To stop enter >stop");
System.out.println(handler);
```
TestEmailHandler - get String

```java
private static String getString(String query) {
    // Use the given String to query the user for an input String and return it.
    System.out.print(query + ">");
    return Keyboard.in.readString();
}
```

TestEmailHandler - abort

```java
private static void abort(EmailHandler handler, String sender, String destination) {
    // Try to abort an Email with the given sender and destination.
    Email email;
    System.out.println("Aborted.");
    email = handler.abort(sender, destination);
    System.out.println(email);
}
```

TestEmailHandler - check

```java
private static void check(EmailHandler handler, String sender, String destination) {
    // Try to check an Email with the given sender and destination.
    System.out.println("Checking for " + "sender: " + sender + " destination: " + destination);
    if (handler.isWaiting(sender, destination))
        System.out.println("FOUND");
    else
        System.out.println("NOT FOUND");
}
```

TestEmailHandler - normal

```java
private static void normal(EmailHandler handler, String sender, String destination) {
    // Add a normal Email with the given sender and destination.
    String body;
    Email email;
    body = TestEmailHandler.getString("Email body:");
    email = new Email(sender, destination, body);
    System.out.println("Adding normal.");
    handler.addNormal(email);
}
```

TestEmailHandler - urgent

```java
private static void urgent(EmailHandler handler, String sender, String destination) {
    // Add an urgent Email with the given sender and destination.
    String body;
    Email email;
    body = TestEmailHandler.getString("Email body:");
    email = new Email(sender, destination, body);
    System.out.println("Adding urgent.");
    handler.addUrgent(email);
}
```

TestEmailHandler - send

```java
private static void send(EmailHandler handler) {
    // Add an Email with the given sender and destination.
    Email email;
    System.out.println("Sending.");
    email = handler.send();
    email = handler.send();
    System.out.println(email);
}
```