**CHAPTER 5**

**Implementation**

### Data Structure

* + 1. **Vectors:**

[*java.lang.Object*](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html)

*extended by*[*java.util.AbstractCollection*](http://java.sun.com/j2se/1.4.2/docs/api/java/util/AbstractCollection.html)

*extended by*[*java.util.AbstractList*](http://java.sun.com/j2se/1.4.2/docs/api/java/util/AbstractList.html)

*extended by****java.util.Vector***

The Vector class implements a growing array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Each vector tries to optimize storage management by maintaining a capacity and a capacity Increment. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacity Increment. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

* + 1. **DISPLAY**

[java.lang.Object](http://java.sun.com/javame/reference/apis/jsr118/java/lang/Object.html)

extended byjavax.microedition.lcdui.Display

Display represents the manager of the display and input devices of the system. It includes methods for retrieving properties of the device and for requesting that objects be displayed on the device. Other methods that deal with device attributes are primarily used with [Canvas](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Canvas.html) objects and are thus defined there instead of here.

There is exactly one instance of Display per [MIDlet](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/midlet/MIDlet.html) and the application can get a reference to that instance by calling the [getDisplay()](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Display.html#getDisplay%28javax.microedition.midlet.MIDlet%29) method. The application may call the getDisplay() method at any time during course of its execution. The Display object returned by all calls to getDisplay() will remain the same during this time.

* + 1. **LOCAL DEVICE**

java.lang.Object

extended by javax.bluetooth.LocalDevice

public class LocalDevice extends java.lang.Object

This class is defined by the JSR-82 specification Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.

This class is used to obtain a singleton instance of the host device.

* + 1. **REMOTE DEVICE**

java.lang.Object

extended by javax.bluetooth.RemoteDevice

public class RemoteDevice extends java.lang.Object

This class is defined by the JSR-82 specification Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.

This class is used to store the instance of the Remote Device discovered.

* + 1. **DISCOVERY AGENT**

java.lang.Object

extended by javax.bluetooth.DiscoveryAgent

public class DiscoveryAgent extends java.lang.Object

This class is defined by the JSR-82 specification Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.

Performs device discovery and service search for the local device.

* + 1. **FORM**

[java.lang.Object](http://java.sun.com/javame/reference/apis/jsr118/java/lang/Object.html)

extended by[javax.microedition.lcdui.Displayable](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Displayable.html)

extended by[javax.microedition.lcdui.Screen](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Screen.html)

extended byjavax.microedition.lcdui.Form

public class Formextends [Screen](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Screen.html)

A Form is a Screen that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items. In general, any subclass of the [Item](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/Item.html) class may be contained within a form. The implementation handles layout, traversal, and scrolling. The entire contents of the Form scrolls together.

* + 1. **COMMAND**

[java.lang.Object](http://java.sun.com/javame/reference/apis/jsr118/java/lang/Object.html)

extended byjavax.microedition.lcdui.Command

public class Commandextends [Object](http://java.sun.com/javame/reference/apis/jsr118/java/lang/Object.html)

The Command class is a construct that encapsulates the semantic information of an action. The behavior that the command activates is not encapsulated in this object. This means that command contains only information about "command" not the actual action that happens when command is activated. The action is defined in a [CommandListener](http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/lcdui/CommandListener.html) associated with the Displayable. Command objects are presented in the user interface and the way they are presented may depend on the semantic information contained within the command.

Commands may be implemented in any user interface construct that has semantics for activating a single action. This, for example, can be a soft button, item in a menu, or some other direct user interface construct.

### 5.2 Module Definition

#### 5.2.1 ProviderMidlet

**Class ProviderMidlet**

* This MIDlet is used to start the Cab Driver side application using the StartApp() method.
* It calls the Setup class.
* Also DestroyApp(boolean) is called to exit the application from the same MIDlet.

**Class Setup**

* This class is the heart of the application.
* It polls for incoming connections needed for the Application.
* Data structures: Vectors to store connections, Connection objects, Choicegroup for displaying the list, LocalDevice for managing host device, Commands.
* Run (): Used to listen for incoming connections and store them in a Vector.
* commandAction (): To handle commands selected by the user.

**Class DisplayList**

* Used for displaying the list of customers (Destination, Pickup Point, Mobile number).
* Data structures: Vectors to store connections, Connection objects, Choicegroup for displaying the list and commands.
* receive (): This is used to receive the Destination, Pickup Point, Mobile number from the connections in the Vector.
* GenerateList (): This generates the list of customers for the driver to select one among them.
* negativeAck (): Sends negative acknowledgment to others i.e. no message to all the other connections except the selected one.
* commandAction (): To handle commands selected by the user.

**Class AcknowledgeForm**

* If the customer accepts the confirmation then this form is used to confirm the deal.
* Data structures: Connection objects, Choicegroup for displaying the list and commands.
* SendConfirmation (): Send a yes or no as message depending on the Choicegroup element selected.
* commandAction (): To handle commands selected by the user.

**Class Finalize**

* Finalize receives passcode and time of meeting from the user and terminates the application.
* Data structures: Connection objects and commands.
* commandAction (): To handle commands selected by the user.

#### 5.2.2 User

**Class UserMidlet**

* This MIDlet is used to start the Cab Driver side application using the StartApp() method.
* It calls the Setup class.
* Also DestroyApp(boolean) is called to exit the application from the same MIDlet.

**Class Setup**

* This class is the heart of the application.
* It performs Device Discovery and Service Search and stores the devices and connections in the Vectors.
* Data structures: Vectors to store connections, Connection objects, Choicegroup for displaying the list, LocalDevice for managing host device, RemoteDevice for discovered devices and Commands.
* Run (): Initialize the localdevice and start device discovery
* deviceDiscovered (): Called every time a device is discovered (Abstract).
* inquiryCompleted (): Called when all devices have been discovered (Abstract).
* Servicediscovered (): Called each time a matching service is discovered (Abstract).
* serviceSearchCompleted (): Called when all services have been discovered (Abstract).
* commandAction (): To handle commands selected by the user.

**Class EntryForm**

* This class is used to obtain parameters (Destination, Pickup Point, Mobile number) and send to all connections in the vector.
* Data structures: Vectors to store connections, Connection objects, Textfields for input and commands.
* receive (): Receive confirmation status from the connections and create a list of cab drivers who have confirmed 1 - yes, 0 - no.
* EstablishConnections(): Send the parameters to all the connections in the vector.
* commandAction (): To handle commands selected by the user.

**Class SelectForm**

* Display and Select one from the cab driver list who have confirmed with a yes or no.
* Data structures: Vectors to store connections, Connection objects, Choicegroup for displaying the list and commands.
* GenerateList(): Displays a list of cabbies who have confirmed to the request.
* commandAction (): To handle commands selected by the user.

**Class Finalize**

* Used to Send 4 digit passcode and meeting time which is used for verification at the time
* Data structures: Connection objects, Textfields for input and commands.
* Finish (): To send the Passcode and meeting time to the selected connection.
* commandAction (): To handle commands selected by the user.

**5.3 COMMON CASE SCENARIO**

**Start Provider App on Cab Providers phone**

**Stop**

ProviderMidlet->StartApp()

Setup->t.run()

DisplayList->receive()

DisplayList- >GenerateList()

AcknowledgementForm>SendConfirmation()

Finalize->Constructor

Connections are sent from User app to Requester App. Both use vectors to store connections.

Send input details to all the connections

Create a list of Acknowledged connections

Send Passcode and time to the selected list

UserMidlet -> StartApp()

Setup -> t.run()

EntryForm -> EstablishConnections()

EntryForm -> receive()

SelectForm -> GenerateList()

Finalize -> Finish()

**Start User App on Cab Requesters Phone**

**Stop**

Fig 5.1 Commom Case Scenario