Experiment No. 10

Name-Rebecca Dias Roll No-19 BECMPN A PID-182027 Batch- 2

AIM: To implement Mobile node discovery in J2ME.

THEORY:

The mobile node is an end system or device such as a cell phone, PDA (Personal Digital assistant), or laptop whose software enables network roaming capabilities. A mobile node is an Internet-connected device whose location and point of attachment to the Internet may frequently be changed. This kind of node is often a cellular telephone or handheld or laptop computer, although a mobile node can also be a router. Special support is required to maintain Internet connections for a mobile node as it moves from one network or subnet to another, because traditional Internet routing assumes a device will always have the same IP address. Therefore, using standard routing procedures, a mobile user would have to change the device's IP address each time they connected through another network or subnet.

Since mobility and ease of connection are crucial considerations for mobile device users, organizations that want to promote mobile communications are putting a great deal of effort into making mobile connection and uncomplicated for the user. The Internet Engineering Task Force (IETF) Mobile IP working group has developed several standards or proposed standards to address these needs, including Mobile IP and later enhancements, Mobile IP version 6 (MIPv6) and Hierarchical Mobile IP version 6 (HMIPv6).

Agent Discovery

During the agent discovery phase the HA and FA advertise their services on the network by using the ICMP router discovery protocol (IROP). Mobile IP defines two methods: agent advertisement and agent solicitation which are in fact router discovery methods plus extensions. Agent advertisement: For the first method, FA and HA advertise their presence periodically using special agent advertisement messages. These messages advertisement can be seen as a beacon broadcast into the subnet. For this advertisement internet control message protocol (ICMP) messages according to RFC 1256, are used with some mobility extensions. Agent solicitation: If no agent advertisements are present or the inter arrival time is too high, and an MN has not received a COA, the mobile node must send agent solicitations. These solicitations are again bases on RFC 1256 for router solicitations.

CODE:

discover_device.java package
mypackage; import
java.io.IOException; import
javax.bluetooth.*;
import javax.bluetooth.DiscoveryListener; import
javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class discover_device extends MIDlet implements CommandListener,DiscoveryListener {
private final List deviceList;
private final Command Exit,Refresh; private
String deviceName;
private DiscoveryAgent agent; private

```
Alert dialog;
public discover_device()

{
deviceList = new List("List of Devices",List.IMPLICIT); Exit= new Command("Exit",Command.EXIT, 0);
Refresh = new Command("Refresh",Command.SCREEN, 1); deviceList.addCommand(Exit);
deviceList.addCommand(Refresh); deviceList.setCommandListener(this);
Display.getDisplay(this).setCurrent(deviceList); }
public void startApp() { try {
deviceList.deleteAll();
LocalDevice local = LocalDevice.getLocalDevice(); local.setDiscoverable(DiscoveryAgent.GIAC);
deviceName = local.getFriendlyName(); agent = local.getDiscoveryAgent();
}
catch (BluetoothStateException ex) {
ex.printStackTrace();
}
try {
agent.startInquiry(DiscoveryAgent.GIAC, this);
}
catch (BluetoothStateException ex) {
ex.printStackTrace();
}}

public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}
public void commandAction(Command c, Displayable d) {
if(c==Exit)
{
this.destroyApp(true);
notifyDestroyed();
}
if(c==Refresh){
this.startApp();
}}
 public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
String deviceaddress = null;
try {
deviceaddress = btDevice.getBluetoothAddress();//btDevice.getFriendlyName(true);
} catch (Exception ex) { ex.printStackTrace();
}
deviceList.insert(0, deviceaddress , null);
}
public void servicesDiscovered(int transID, ServiceRecord[] servRecord) { throw new
UnsupportedOperationException("Not supported yet."); }
public void serviceSearchCompleted(int transID, int respCode) { throw new
UnsupportedOperationException("Not supported yet.");
}
public void inquiryCompleted(int discType) { Alert
dialog = null;

if (discType != DiscoveryListener.INQUIRY_COMPLETED) {
```

```
dialog = new Alert("Bluetooth Error","The inquiry failed to complete normally",null,
AlertType.ERROR);
}
else {
dialog = new Alert("Inquiry Completed","The inquiry completed normally", null,AlertType.INFO);
}
dialog.setTimeout(500);
Display.getDisplay(this).setCurrent(dialog);
}}
```

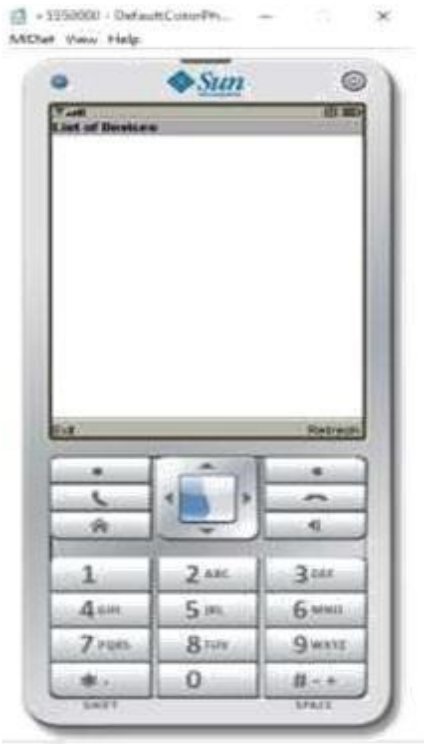
Blue.java

```
package mypackage; import
java.io.*;
import javax.microedition.midlet.*; import
javax.microedition.lcdui.*; import
javax.microedition.io.*; import
javax.bluetooth.*;
import java.util.*;
public class Blue extends MIDlet implements CommandListener,DiscoveryListener
{private List activeDevices; private
Command select,exit; private Display
display;
private LocalDevice local=null; private
DiscoveryAgent agent = null; private Vector
devicesFound = null;
private ServiceRecord[] servicesFound = null; private
String connectionURL = null;
public void startApp() {
display = Display.getDisplay(this);
activeDevices = new List("Active Devices", List.IMPLICIT); select = new Command("Search Again",
Command.OK, 0); exit = new Command("Exit", Command.EXIT, 0); activeDevices.addCommand(exit);
activeDevices.setCommandListener(this); try {
local = LocalDevice.getLocalDevice(); } catch
(Exception e) {} doDeviceDiscovery();
display.setCurrent(activeDevices);} public void
pauseApp() {}
public void destroyApp(boolean unconditional) { notifyDestroyed(); } public void
commandAction(Command cmd, Displayable disp) {
if (cmd == select && disp == activeDevices) {
activeDevices.deleteAll(); doDeviceDiscovery();}
if (cmd == exit) { destroyApp(false); }} public void
inquiryCompleted(int param) { try {switch (param)
{
case DiscoveryListener.INQUIRY_COMPLETED:
if (devicesFound.size() > 0) { activeDevices.addCommand(select);
activeDevices.setSelectCommand(select);}
else { activeDevices.append("No Devices Found", null); } break; }
}
catch (Exception e) {}
}
public void serviceSearchCompleted(int transID, int respCode) {}
```

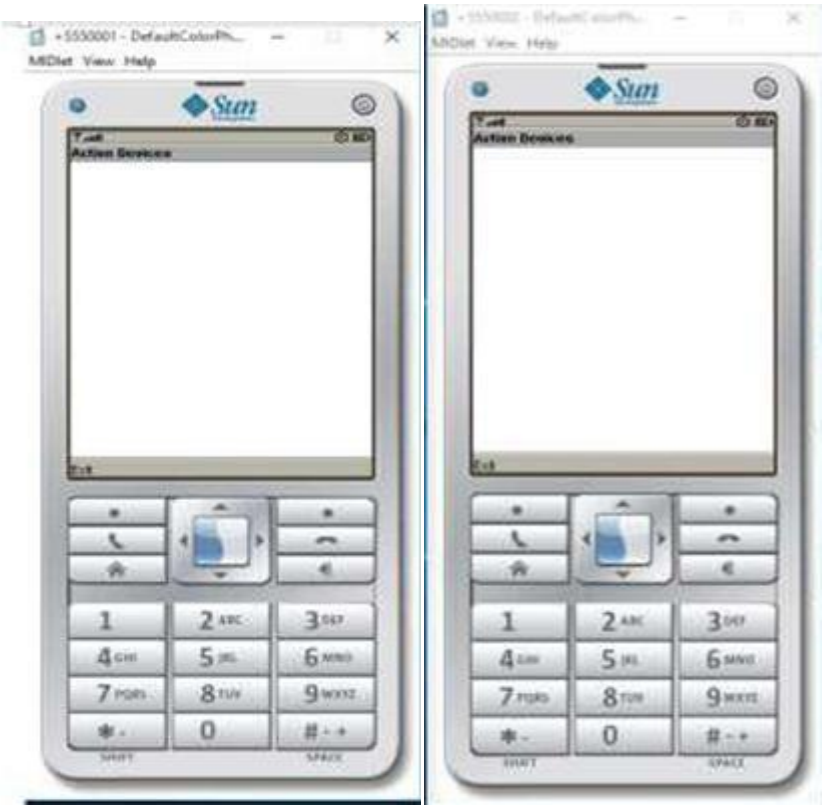
```
public void servicesDiscovered(int transID, ServiceRecord[] serviceRecord) {}
public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass)
{String str = null; try {
str = remoteDevice.getBluetoothAddress() + " - "; str += remoteDevice.getFriendlyName(true);
} catch (Exception e) {} activeDevices.append(str, null); devicesFound.addElement(remoteDevice); try
{
if (!agent.startInquiry(DiscoveryAgent.GIAC, this)) {} } catch (BluetoothStateException e) {
//          TODO Auto-generated catch block e.printStackTrace();}
}
private void doDeviceDiscovery() {
try {local = LocalDevice.getLocalDevice(); agent =
local.getDiscoveryAgent(); devicesFound = new
Vector();
} catch (Exception e) {}}
}
```

OUTPUT:

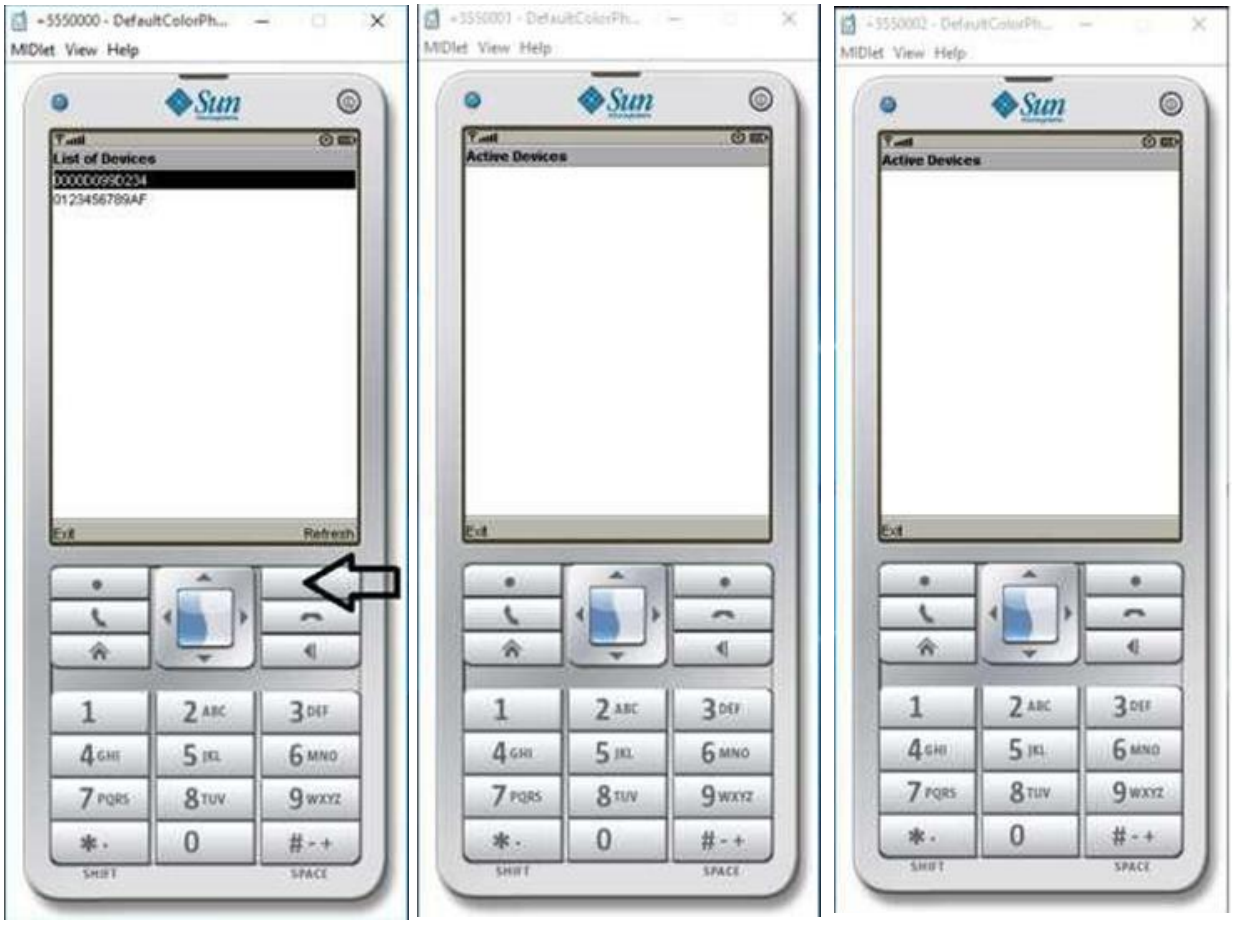
Step 1: First run discover_device.java. You will see this as the output. **(Hereafter referred as Device 0)**



Step 2: Run Blue.java file (You can run this file any number of times). For example, I have run this file for two times. So here I can see that two devices are running. (Hereafter referred as Device 1 and Device 2 respectively).



Step 3: Click in Device 0 to the button pointed by the arrow. After clicking you will be able to see the Device number of Device 1 & Device 2 as shown below.



CONCLUSION:

From this experiment, we learnt about mobile node discovery. A mobile node uses method called agent discovery which determines information like when the node has moved from one location to another or whether the network is the node's home or a foreign network. We implemented the same using Java and observed the final output.