Sub: J2ME.

Q-1 Answer any four 20 Marks

Q-A) Lifecycle of a MIDlet

Ans:

The AMS controls the transition from one MIDlet state to another by using the startApp(), pauseApp(), and destroyApp() methods of the Applet class.

When you run a MIDlet, the AMS creates an instance of the MIDlet by using the MIDlet constructor, and the MIDlet enters the paused state.

In the paused state, the AMS invokes the startApp() method to start the MIDlet, and the MIDlet enters the active state. In this state, the MIDlet acquires the resources it needs to begin its services in the active state.

If the AMS identifies that the active MIDlet is not currently required, the AMS invokes the pauseApp() method to pause the running MIDlet. When the MIDlet reaches the paused state, it releases all the resources that it has acquired in the active state.

The MIDlet can re-enter the active state if the AMS calls the startApp() method again.

When the AMS identifies that the active MIDlet is no longer required and a high priority MIDlet is waiting to be run, then the AMS destroys the active MIDlet by calling the destroyApp() method. The MIDlet releases all the resources before entering the destroyed state.

An active MIDlet can send a request to the AMS to prevent its entry in the destroyed state by throwing the MIDletStateChangeException. This occurs when the MIDlet is in the middle of an important process. However, AMS may or may not grant this request. If the unconditional property of the destroyApp() method is set to true, the AMS ignores the request from the MIDlet. However, the request is granted if the unconditional property is set to false and the destroyApp() method is called after some time.

A MIDlet can also voluntarily notify the AMS about a requested state that the MIDlet wants to enter. In this condition, the AMS will not call the pauseApp() or the destroyApp() method. If the MIDlet wants to enter the paused state or the destroyed state, it can call the notifyPause() method or the notifyDestroy() method. The MIDlet needs to perform all resource cleaning activities before notifying the AMS.

A MIDlet can re-enter the active state only from the paused state. However, it can enter the destroyed state from both the paused state and the active state. A MIDlet can enter the paused state either from the active state or when the AMS initially launches the MIDlet.

The following syntax shows the structure of a MIDlet class:

```
public class MyMIDlet extends MIDlet
{

        // Defining the default constructor of MIDlet class.
        MyMIDlet( )
        {
        }
        protected void startApp( ) throws MIDletStateChangedException
        {
        }

        protected void pauseApp( )
        {
        }

    protected void destroyApp(boolean unconditional) throws MIDletStateChangedException
        {
        }
}
```

Q1. B) J2ME Architecture

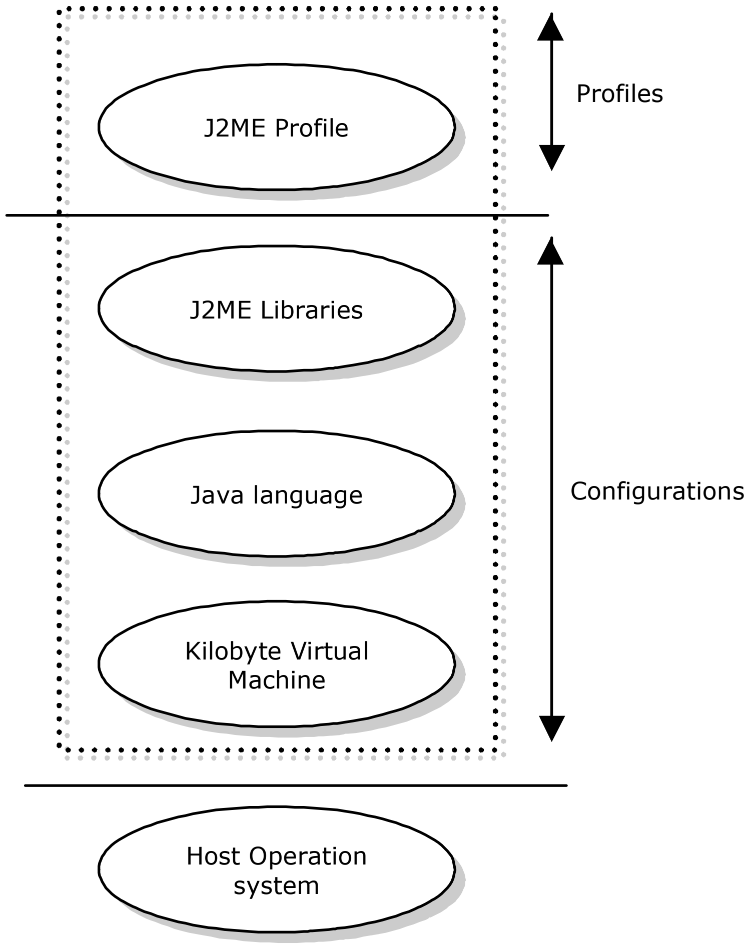
Ans:

The J2ME architecture is designed for small devices that have common hardware and software characteristics. These characteristics are:

- Limited memory, ranging from 128 KB to 2 MB.
- Limited computational capability. These devices have either 16-bit or 32-bit processors.

The J2ME architecture consists of the following main components:

- Configuration
- Profile



The following figure shows the J2ME architecture:

Components of J2ME

A J2ME configuration specifies a minimum Java platform for small devices having similar memory constraints and processing capability. For example, a configuration can define a Java platform for all handheld devices, such as mobile phones and PDAs having similar hardware resources in terms of memory and processor. A J2ME configuration comprises a virtual machine, Java language features, and minimum class libraries that are designed to support such small devices.

The configuration supported by J2ME for wireless mobile devices is called Connected Limited Device Configuration (CLDC). The devices that support CLDC have a simple user interface and low hardware resources.

Q1. D) <u>Screen and Its Elements</u>

Ans:

The Screen is a subclass of the Displayable class. It is used to display the objects on the screen of a mobile device. The Screen is an abstract class that is used to derive various types of screen classes, which can be used in the UI of mobile applications.

The four classes, which are derived from the Screen class, are:

- List: Creates a screen containing a list of elements. A user can scroll through a list of elements and select them. For example, the phonebook in a mobile phone is a List class object as it contains the list of phonebook entries.
- TextBox: Displays a text box on the screen of the mobile device. You can enter and edit the text displayed in the text box. For example, the SMS compose window of a mobile phone is a TextBox class object.
- Alert: Displays the data to the user for a specified time and then displays the next Displayable object. An Alert object consists of a text string and an image. You can use the Alert objects to notify the users about errors and exceptions occurring in the MIDlet. For example, you can create an Alert object to notify the users about deletion of a SMS message from the inbox of a mobile phone.
- Form: Displays a form containing items, such as text fields, images, and lists of items. A Form object can display several instances of item classes, such as Image, TextField, and ChoiceGroup. For example, you can create a registration form, which consists of text boxes and radio buttons to enable the users to fill their details.

Q1. E) <u>Implementing Form</u>

Ans:

user can interact with a Form class object by modifying the contents of the items displayed on it. For example, the user can select the items from a list or type the text in the text field item of a Form.

The Form class is derived from the Screen class, and contains the following two constructors:

- Form(String title): Creates a blank Form object with no component.
- Form(String title,Item[] items): Creates a new Form object with the specified items. The items to be displayed are passed to this method as an array of Item objects.

The Form class also contains methods to add, remove, and traverse the elements in a Form.

The following table lists the methods defined by the Form class:

Method	***Description***
append(Image imageobject)	Adds an Image object to the current Form object. This method returns the index number of newly added Image object.
append(Item itemname)	Adds an Item object to the current Form object. This method returns the index number of the newly added Item object.
append(String stringobject)	Adds a textual string to the current Form object. The string is passed as an argument of String object, String. This method returns the index number of the newly added string.
insert(int itemNumber, Item item)	Adds the Item object to the current form before the item with the index number, itemNumber.
delete(int itemNumber)	Deletes the item referenced by the index, itemNumber.
DeleteAll()	Deletes the items from the current Form.

Method	***Description***
get(int itemNumber)	Returns the item, which has the index number, itemNumber.
getHeight()	Returns the height of the display area, which is available for displaying items.
getWidth()	Returns the width of the display area, which is available for displaying items.
set(int itemNumber, Item itemname)	Replaces the Item object represented by the specified itemNumber, with the itemname.
size()	Returns the number of items in the current Form.

Q2. Answer any four in details: 20 Marks