CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 3 – Nov. 2019

Sub:	Mobile Applications							Sub Code:	17MCA53
Date:	18/11//2019	Duration:	90 min's	Max Marks:	50	Sem:	V	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
	PART I			
1	Define service. How do you create your own service in Android? Explain with a snippet code.	[10]	CO4	L1,L2
	OR			
2	What is the process for binding activities to services in android application?	[10]	CO5	L2
	PART II			
3	Explain the concepts of networking in terms of communicating between service and activity.	[10]	CO4	L2
	OR			
4	What is a program level in iOS? Discuss the various program levels available for an iOS develop.	[5]	CO4	L1
	PART III			
5	With a neat diagram, explain iOS project.	[10]	CO5	L2
	OR			
6	Describe the anatomy of a windows phone 7 App.	[10]	CO4	L2
	PART IV			
7	Write a note on other useful windows phone Thing.	[10]	CO4	L2
	OR			
8	How to build derby app in windows phone 7? Explain.	[10]	CO5	L3
	PART V			
9	Implement an application that creates an alert upon receiving a message.	[10]	CO2	L3
	OR			
10	Create an application that writes data to the SD card.	[10]	CO2	L3

CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 3– Nov. 2019

Sub:	Mobile Applications					**Sub Code:**	17MCA53	**Branch:**	MCA
Date:	18/11/2019	**Duration:**	90 min's	**Max Marks:**	50	**Sem**	V		OBE

1. Define service. How do you create your own service in Android? Explain with a snippet code.

A service is an application in Android that runs in the background without needing to interact with the user. For example, while using an application, you may want to play some background music at the same time. In this case, the code that is playing the background music has no need to interact with the user, and hence it can be run as a service. Services are also ideal for situations in which there is no need to present a UI to the user.
Following are the steps involved in creating own service.

- Create a new android application.
- Add a new class file to the project and name it MyService.java. Write the following code in it:

```
package net.learn2develop.Services;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;
public class MyService extends Service
{
@Override
public IBinder onBind(Intent arg0)
{
return null;
}
public int onStartCommand(Intent intent, int flags, int startId)
{
Toast.makeText(this,"ServiceStarted",Toast.LENGTH_LONG).show();
return START_STICKY;
}
public void onDestroy()
{
super.onDestroy();
Toast.makeText(this,"ServiceDestroyed",Toast.LENGTH_LONG).show();
}
}
```

The onBind() method enables you to bind an activity to a service. This in turn enables an activity to directly access members and methods inside a service. The onStartCommand() method is called when you start the service explicitly using the startService() method. This method signifies the start of the service, and you code it to do the things you need to do for your service. In this method, you returned the constant START_STICKY so that the service will continue to run until it is explicitly stopped. The onDestroy() method is called when the service is stopped using the stopService() method. This is where you clean up the resources used by your

service.
 In the AndroidManifest.xml file, add the following statement :
<service android:name=".MyService" />
 In the activity_main.xml file, add the following statements in bold:
<Button android:id="@+id/btnStartService"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Start Service" />
<Button android:id="@+id/btnStopService"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Stop Service" />
 Add the following statements in bold to the MainActivity.java file:

```
import android.content.Intent;
import android.view.View;
import android.widget.Button;
public class MainActivity extends Activity
{
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Button btnStart = (Button) findViewById(R.id.btnStartService);
btnStart.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
{
startService(new Intent(getBaseContext(), MyService.class));
}
});
Button btnStop = (Button) findViewById(R.id.btnStopService);
btnStop.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
{
stopService(new Intent(getBaseContext(), MyService.class));
}
});
}
}
```

2.What is the process for binding activities to services in android application?

Often a service simply executes in its own thread, independently of the activity that calls it. This doesn't pose any problem if you simply want the service to perform some tasks periodically and the activity does not need to be notified of the status of the service. For example, you may have a service that periodically logs the geographical location of the device to a database. In this case, there is no need for your service to interact with any activities, because its main purpose is to save the coordinates into a database. However, suppose you want to monitor for a particular location. When the service logs an address that is near the location you are monitoring, it might need to communicate that information to the activity. In this case, you would need to devise a way for the service to interact with the activity.

BINDING ACTIVITIES TO SERVICES

Real-world services are usually more sophisticated, requiring the passing of data so that they can do the job correctly for you. Using the service demonstrated earlier that

downloads a set of files, suppose you now want to let the calling activity determine what fi
les to download, instead of hardcoding them in the service. Here is what you need to do.
First, in the calling activity, you create an Intent object, specifying the service name:

```
Button btnStart = (Button) findViewById(R.id.btnStartService);
btnStart.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
{
Intent intent = new Intent(getBaseContext(), MyService.class);
}
});
```

You then create an array of URL objects and assign it to the Intent object through its
putExtra() method. Finally, you start the service using the Intent object:

```
Button btnStart = (Button) findViewById(R.id.btnStartService);
btnStart.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
{
Intent intent = new Intent(getBaseContext(), MyService.class);
try
{
URL[] urls = new URL[]
{
new URL("http://www.amazon.com/somefiles.pdf"),
new URL("http://www.wrox.com/somefiles.pdf"),
new URL("http://www.google.com/somefiles.pdf"),
new URL("http://www.learn2develop.net/somefiles.pdf")
};
intent.putExtra("URLs", urls);
} catch (MalformedURLException e)
{
e.printStackTrace();
}
startService(intent);
}
});
```

Note that the URL array is assigned to the Intent object as an Object array. On the
service's end, you need to extract the data passed in through the Intent object in the
onStartCommand() method:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
// We want this service to continue running until it is explicitly
// stopped, so return sticky.
Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
Object[] objUrls = (Object[]) intent.getExtras().get("URLs");
URL[] urls = new URL[objUrls.length];
for (int i=0; i<objUrls.length-1; i++)
{
urls[i] = (URL) objUrls[i];
}
new DoBackgroundTask().execute(urls);
return START_STICKY;
}
```

The preceding first extracts the data using the getExtras() method to return a Bundle
object. It then uses the get() method to extract out the URL array as an Object array.
Because in Java you cannot directly cast an array from one type to another, you have to

create a loop and cast each member of the array individually. Finally, you execute the background task by passing the URL array into the execute() method.
This is one way in which your activity can pass values to the service. As you can see, if you have relatively complex data to pass to the service, you have to do some additional work to ensure that the data is passed correctly. A better way to pass data is to bind the activity directly to the service so that the activity can call any public members and methods on the service directly.

3.Explain the concepts of networking in terms of communicating between service and activity.
One can communicate with the external world via SMS, email or using HTTP protocol. Using the HTTP protocol, you can perform a wide variety of tasks, such as downloading web pages from a web server, downloading binary data, and so on. The following project creates an Android project so that you can use the HTTP protocol to connect to the Web to download all sorts of data.
 Create a new android project and name it as Networking
 Add the following line in AndroidManifest.xml file:
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>
 Import the following namespaces in the MainActivity.java file:

```
package net.learn2develop.Networking;
import android.app.Activity;
import android.os.Bundle;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.widget.ImageView;
import android.widget.Toast;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

 Define the OpenHttpConnection() method in the MainActivity.java file:

```
public class MainActivity extends Activity
{
private InputStream OpenHttpConnection(String urlString)
throws IOException
{
InputStream in = null;
int response = -1;
URL url = new URL(urlString);
URLConnection conn = url.openConnection();
if (!(conn instanceof HttpURLConnection))
throw new IOException("Not an HTTP connection");
try
{
HttpURLConnection httpConn = (HttpURLConnection) conn;
httpConn.setAllowUserInteraction(false);
```

```
httpConn.setInstanceFollowRedirects(true);
httpConn.setRequestMethod("GET");
httpConn.connect();
response = httpConn.getResponseCode();
if (response == HttpURLConnection.HTTP_OK)
{
in = httpConn.getInputStream();
}
}
catch (Exception ex)
{
throw new IOException("Error connecting");
}
return in;
}
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
}
```

 Run the application to establish a connection.

 Working Procedure: Because you are using the HTTP protocol to connect to the Web, your application needs the INTERNET permission; hence, the first thing you do is add the permission in the AndroidManifest.xml file. You then define the OpenHttpConnection() method, which takes a URL string and returns an InputStream object. Using an InputStream object, you can download the data by reading bytes from the stream object. In this method, you made use of the HttpURLConnection object to open an HTTP connection with a remote URL. You set all the various properties of the connection, such as the request method, and so on. After you try to establish a connection with the server, you get the HTTP response code from it. If the connection is established (via the response code HTTP_OK), then you proceed to get an InputStream object from the connection. Using the InputStream object, you can then start to download the data from the server.

Communication between Services and Activities

Real-world services are usually more sophisticated, requiring the passing of data so that they can do the job correctly for you. Using the service demonstrated earlier that downloads a set of files, suppose you now want to let the calling activity determine what fi les to download, instead of hardcoding them in the service. Here is what you need to do.

First, in the calling activity, you create an Intent object, specifying the service name:

```
Button btnStart = (Button) findViewById(R.id.btnStartService);
btnStart.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
{
Intent intent = new Intent(getBaseContext(), MyService.class);
}
});
```

You then create an array of URL objects and assign it to the Intent object through its putExtra() method. Finally, you start the service using the Intent object:

```
Button btnStart = (Button) findViewById(R.id.btnStartService);
btnStart.setOnClickListener(new View.OnClickListener()
{
public void onClick(View v)
```

```
{
Intent intent = new Intent(getBaseContext(), MyService.class);
try
{
URL[] urls = new URL[]
{
new URL("http://www.amazon.com/somefiles.pdf"),
new URL("http://www.wrox.com/somefiles.pdf"),
new URL("http://www.google.com/somefiles.pdf"),
new URL("http://www.learn2develop.net/somefiles.pdf")
};
intent.putExtra("URLs", urls);
} catch (MalformedURLException e)
{
e.printStackTrace();
}
startService(intent);
}
});
```

Note that the URL array is assigned to the Intent object as an Object array. On the service's end, you need to extract the data passed in through the Intent object in the onStartCommand() method:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
// We want this service to continue running until it is explicitly
// stopped, so return sticky.
Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
Object[] objUrls = (Object[]) intent.getExtras().get("URLs");
URL[] urls = new URL[objUrls.length];
for (int i=0; i<objUrls.length-1; i++)
{
urls[i] = (URL) objUrls[i];
}
new DoBackgroundTask().execute(urls);
return START_STICKY;
}
```

The preceding first extracts the data using the getExtras() method to return a Bundle object. It then uses the get() method to extract out the URL array as an Object array. Because in Java you cannot directly cast an array from one type to another, you have to create a loop and cast each member of the array individually. Finally, you execute the background task by passing the URL array into the execute() method.

This is one way in which your activity can pass values to the service. As you can see, if you have relatively complex data to pass to the service, you have to do some additional work to ensure that the data is passed correctly. A better way to pass data is to bind the activity directly to the service so that the activity can call any public members and methods on the service directly.

4. What is a program level in iOS? Discuss the various program levels available for an iOS development

The program level is where development teams, stakeholders, and other resources are devoted to some important, ongoing system development mission. It describes the program level teams, roles, and activities that incrementally deliver a continuous flow of value.

- **iOS Developer Program**

This program level allows developers to distribute apps in the App Store as an individual, a sole proprietor, a company, an organization, a government entity, or an educational institution. The cost for this program is $99 a year.

- **iOS Developer Enterprise Program**

This program level allows developers to develop proprietary apps for internal distribution within your company, organization, government entity, or educational institution. The cost for this program is $299 a year.

- **iOS Developer University Program**

This program level allows higher-education institutions to create teams of up to 200 developers that can develop iOS applications. This program level is free, and allows for programs to be tested on physical devices, but does not allow for ad hoc or App Store deployment.

5.With a neat diagram, explain iOS project.

There are many steps to be followed before you start creating an IOS application. Here we will discuss few of them.

Anatomy of an iOS App

The files that are actually deployed to the iOS device are known as .app files and these are just a set of directories. Although there is an actual binary for the iOS application, you can open the .app file and find the images, meta data, and any other resources that are included.

 Views: iPhone apps are made up of one or more views. Views usually have GUI elements such as text fields, labels, buttons, and so on. You can build a view built using the Interface Builder tool, which enables you to drag and drop controls on the view, or you can create a view entirely with code.

 Code that makes the Views work: Because iOS applications follow the MVC design pattern, there is a clean break between the UI and code that provides the application code.

 Resources: Every iOS application contains an icon file, an info.plist file that holds information about the application itself and the binary executable. Other resources such as images, sounds, and video are also classified as resources.

 Project Structure in Depth: When an iOS project is created within xCode, the IDE creates a set of files that are ready to run. These files provide the basics of what is needed to get going with a new project.

o **Main.m:** As with any C program, the execution of Objective-C applications start from the main() function, which is the main.m file.

o **AppDelegate.m:** The AppDelegate receives messages from the application object during the lifetime of your application. The AppDelegate is called from the operating system, and contains events such as the didFinishLaunchingWithOptions, which is an event that iOS would be interested in knowing about.

o **MainStoryBoard.storyboard:** This is where the user interface is created. In past versions of xCode/iOS the user interface was stored within .xib (pronounced NIB) files. Although this method is still supported, Storyboards are a great improvement over .xib files for applications with complex navigation and many views.

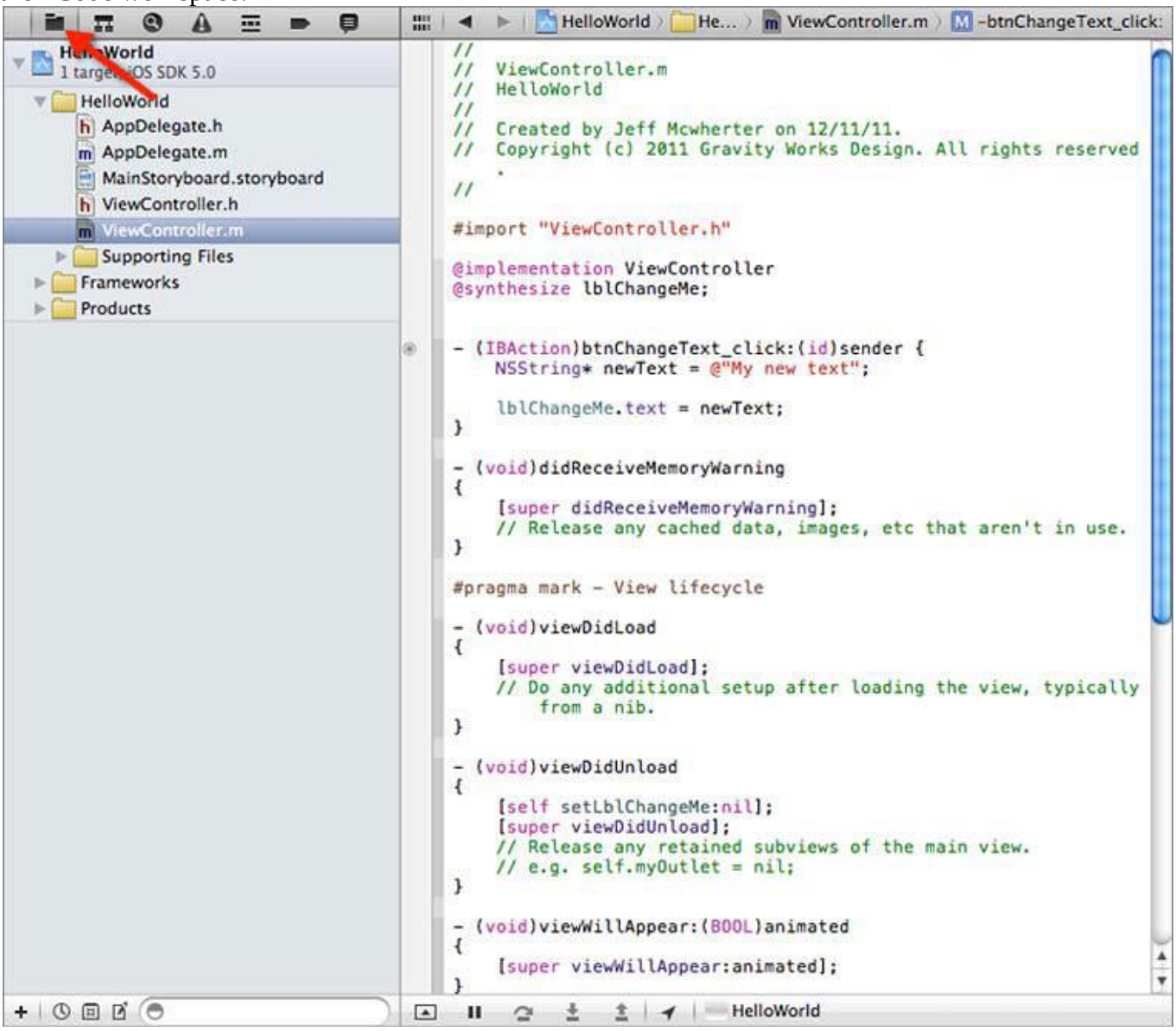
o **Supporting Files:** The supporting files directory contains files such as the plist setting files (which contain customizable application settings), as well as string resource files that are used within your app.

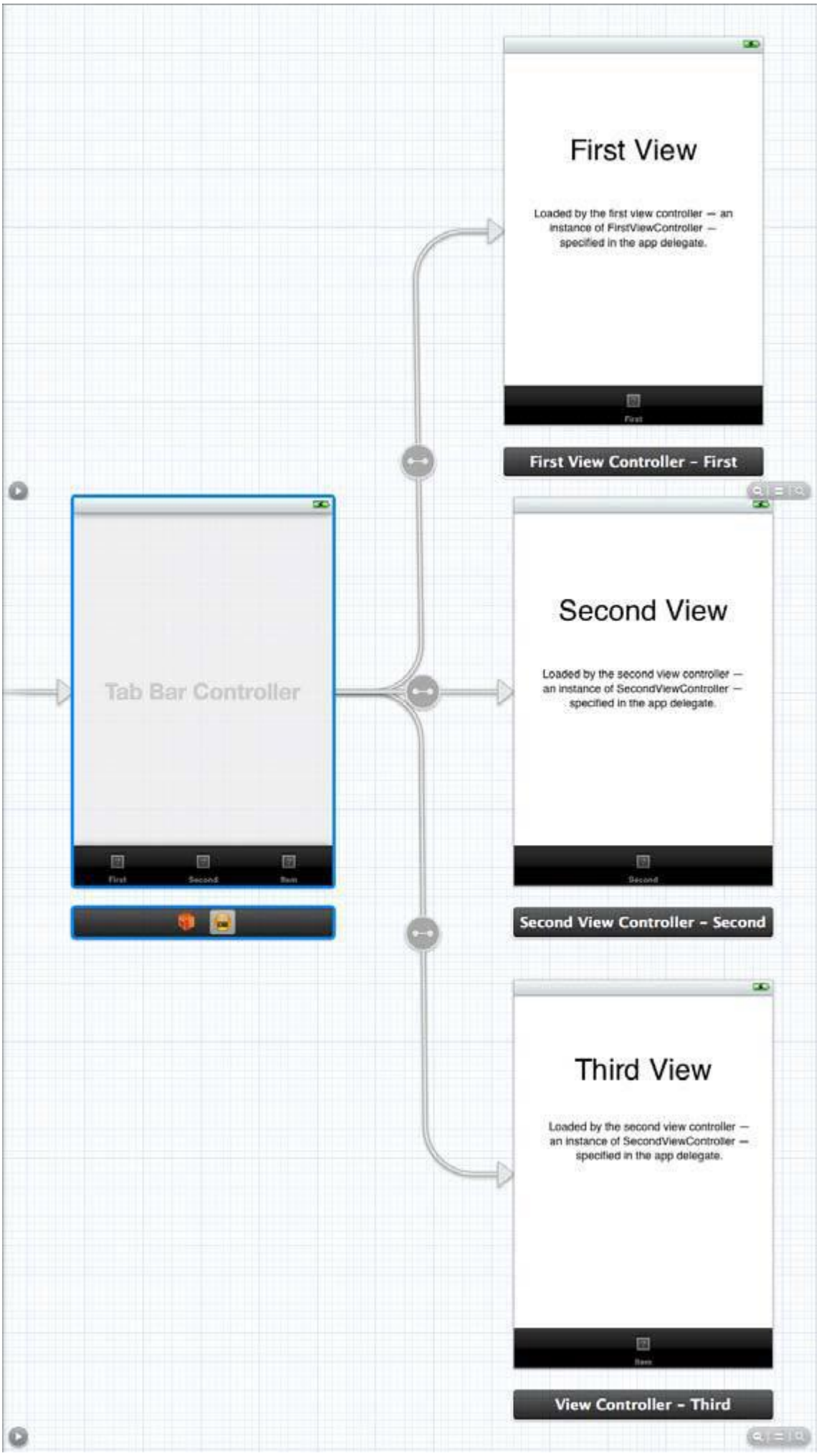
Getting to Know the xCode IDE

It is important to use the correct tool for the job, regardless of whether you are constructing a house or constructing an application. If you are new to xCode, there will be a bit of a learning curve to becoming proficient with the IDE, but xCode is a top-notch IDE with many features for you to discover.

 Navigators: The left side of the xCode window is known as the navigator area. A variety of navigators enable you to list the contents of your project, find errors, search for code, and more. The remainder of this section introduces the Project Navigator, the Search Navigator, and the Issue Navigator. Going from left to right, the project navigator is the first of the xCode navigators; the icon looks like a file folder. The Project Navigator simply shows the contents of your project or workspace, as shown in Figure 5.1. Double-clicking a file in the Project Navigator opens the file in a new window, and single-clicking opens the file within

the xCode workspace.




First View
Loaded by the first view controller — an instance of FirstViewController — specified in the app delegate.
First
First View Controller - First
Tab Bar Controller
First
Second
Item
Second View
Loaded by the second view controller — an instance of SecondViewController — specified in the app delegate.
Second
Second View Controller - Second
Third View
Loaded by the second view controller — an instance of SecondViewController — specified in the app delegate.
Item
View Controller - Third

Storyboards: In iOS versions prior to iOS 5, developers needed to create a separate XIB file for each view of their application. A XIB file is an XML representation of your controls and instance variables that get compiled into the application. Managing an application that contains more than a few views could get cumbersome. iOS 5 contained a new feature called storyboards that enables developers to lay out their workflow using design tools built within xCode. Apps that use navigation and tab bars to transition between views are now much easier to manage, with a visual representation of how the app will flow. With Storyboards, you will have a better conceptual overview of all the views in your app and the connections between them. Figure shows an example of a storyboard for an application containing a tab bar for navigation to three other views.

6. Describe the anatomy of a windows phone 7 App.

Here we discuss the basic design elements used in Windows Phone 7 application development, and how you can leverage the tools you have at hand to implement them.

1. **Storyboards:** Storyboards are Silverlight's control type for managing animations in code. They are defined in a given page's XAML and leveraged using code behind. Uses for these animations are limited only by the transform operations you are allowed to perform on objects.

Anytime you want to provide the user with a custom transition between your pages or element updates, you should consider creating an animation to smooth the user experience.
Because storyboards are held in XAML you can either edit them manually or use Expression Blend's WYSIWYG editor.

In Blend, in the Objects and Timelines Pane at the left, click the (+) icon to create a storyboard. Once you have a storyboard, you can add key frames on your time line for each individual element you would like to perform a transformation on. This can include moving objects and changing properties (like color or opacity). After setting up your time line, you can start the storyboard in code. The name you created for your storyboard will be accessible in code behind.

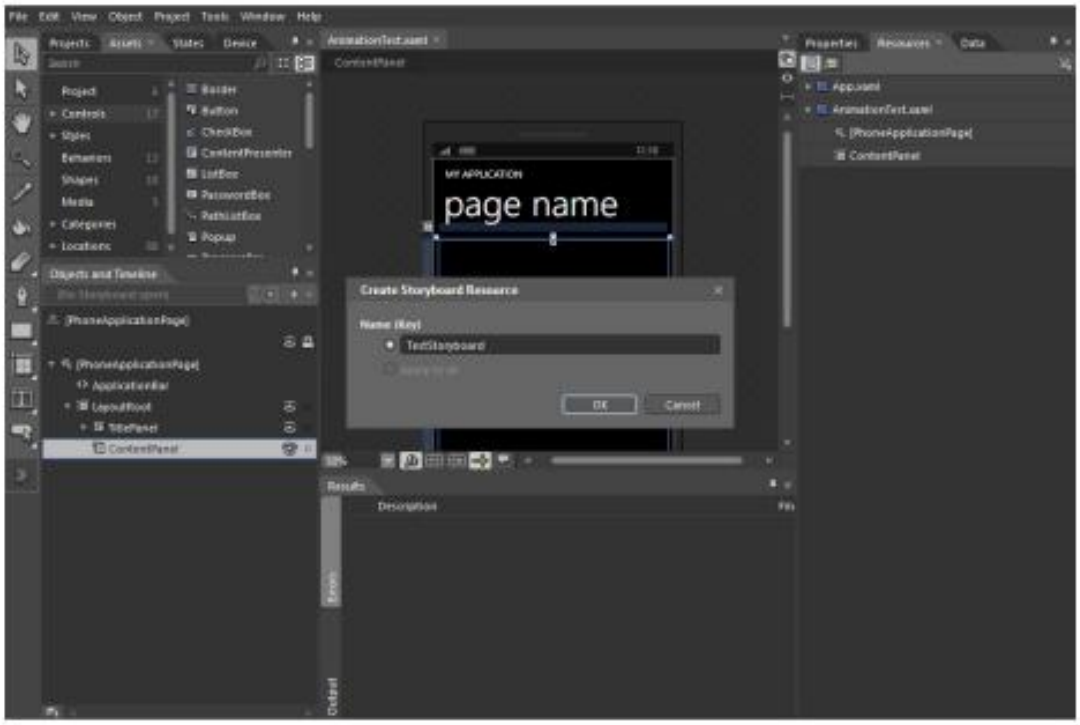


FIGURE 8-9: Storyboards in Blend

2. **Pivot vs Panorama:** Both the Pivot and Panorama controls are used to delineate categories and subsets of data. With the Pivot control you get strict isolation of these groupings, with the menu providing discoverable UI to show the other categories. 2. With the
Panorama control you get transitions between the groupings with discoverable content on the window boundaries.

FIGURE 8-10: Pivot control

FIGURE 8-11: Panorama control

The Windows Phone Emulator

The Windows Phone 7 emulator is a very powerful tool. Not just a simulator, the emulator runs a completely sandboxed virtual machine in order to better mimic the actual device. It also comes with some customization and runtime tools to manipulate sensors that are being emulated on the device, including GPS and accelerometer, as well as provide a way to capture screenshots while testing and developing applications. The debugging experience inside of Visual Studio is superior to the ones in Eclipse and the third-party frameworks. The load time of the Emulator is quite fast. It acts responsively, and the step-through just works.

7.Write a note on other useful windows phone Thing.

IOS applications can be used do many more tasks as explained in following sections.

1. **Offline Storage**

Even if your application is using a web service for retrieving information, at some point you may need to save information on the device. Depending on the size and type of data, you have a few different options.

Plist : Property lists are the simplest way to store information on the device. In the Mac world, many applications use the plist format to store application settings, information about the application, and even serialized objects. It's best to keep the data contained in these files simple and small, though.

The following example finds the path to a plist stored in the supporting files directory, with a name of example. It then loads the plist into a dictionary object, and loops through each time writing the contents of each item in the plist to the debug console.

```
- (void)getValuesFromPlist
{
// build the path to your plist
NSString *path = [[NSBundle mainBundle] pathForResource:
@"example" ofType:@"plist"];
// load the plist into a dictionary
NSDictionary *pListData = [[NSDictionary alloc]
initWithContentsOfFile:path];
// loop through each of the Items in the property list and log
```

```
for (NSString *item in pListData)
NSLog(@"Value=%@", item);
}
```

Core Data: If the data that you need to persist on the device is nontrivial, meaning there is a great deal of it or its complex, Core Data is the way to go. Core Data is described by Apple as a "schemadriven object graph management and persistence framework." Core Data is not an ORM (Object Relational Mapper). Core Data is an API that abstracts the actual data store of the objects. Core Data can be configured to store these objects as a SQLite database, a plist, custom data, or a binary file. Core Data has a steep learning curve, but is well worth learning more about if your app will have a great deal of data held within.

2. GPS

One of the great benefits to mobile devices is the GPS functionality. Once you are able to get over the hurdles of learning the basic functions within the iOS platform, starting to work with the GPS functions can be a great deal of fun. The GPS functions are located in the CoreLocation framework, which is not added to a new project by default. To do this, you will need to click the Build Phases tab on the project settings page.
Once on the Build Phases tab, expand the Link Binary with Libraries section, and click the + button. You are then prompted with a list of frameworks to add. Select the CoreLocation.framework. Use the code given below –

```
// GPS Example
locationManager = [[CLLocationManager alloc] init];
locationManager.delegate = self;
locationManager.distanceFilter = kCLDistanceFilterNone;
// get GPS DatalocationManager.desiredAccuracy =
kCLLocationAccuracyHundredMeters;
[locationManager startUpdatingLocation];
```

Notifications

Setting up notifications for Windows Phone 7 is a multistage process. First you must build up a push channel to receive communications within your app. Creating that push channel provides you with a Service URI to post data to. Posting data in specific formats determines what type of message will be displayed to the client app. There are **three types** of notifications:
1. **Toast notification:** The first and simplest is the *toast notification.* With a toast notification you can pass a title, a string of content, and a parameter.
 The title will be boldfaced then displayed
 the content will follow non-boldfaced, and
• the parameter will not be shown, but it is what is sent to your application when the user taps on the toast message. This can contain parameters to load on the default page, or a relative link to the page you want loaded when the app loads as a result of the tap. Then the user taps on the *toast message.*
2. **Tile notification:** With the tile notification you can update the application tile content. The XML data that you post contains fields for the title on
 the front of the tile,
 front of the tile background image,
 the count for the badge,
 the title for the back of the tile,
 the back of the tile background image, and
 string of content for the back of the tile.
3. **Raw Notifications:** The third and most developer-centric notification type is raw. With the raw notification type you can pass data directly to the app. It will not be delivered if the application is not running.

Accelerometer

In addition to GPS, Windows Phone 7 devices are outfitted with an accelerometer. The emulator provides a 3-D interface for simulating accelerometer change events. You can track the movement of the device by capturing the ReadingChanged event on the accelerometer. However, you need

to have a delegate to call back to the UI thread if you want to display anything special based on the event. If the application can access the UI thread, the ReadingChanged event handler will call the delegate function; otherwise, it will dispatch the event on the UI thread. You must also make sure that when you are done capturing this data, you stop the accelerometer to preserve battery life.

Web Services

The Derby application is an example of leveraging data over the web to add value to your application. If you don't want to be the central repository for all data exposed to your users, you can leverage web services that exist from other vendors.

8. How to build derby app in windows phone 7? Explain.

Here, you implement the features of the Derby Names project using Microsoft Visual Studio, while also taking time to learn Windows Phone 7–specific technologies.

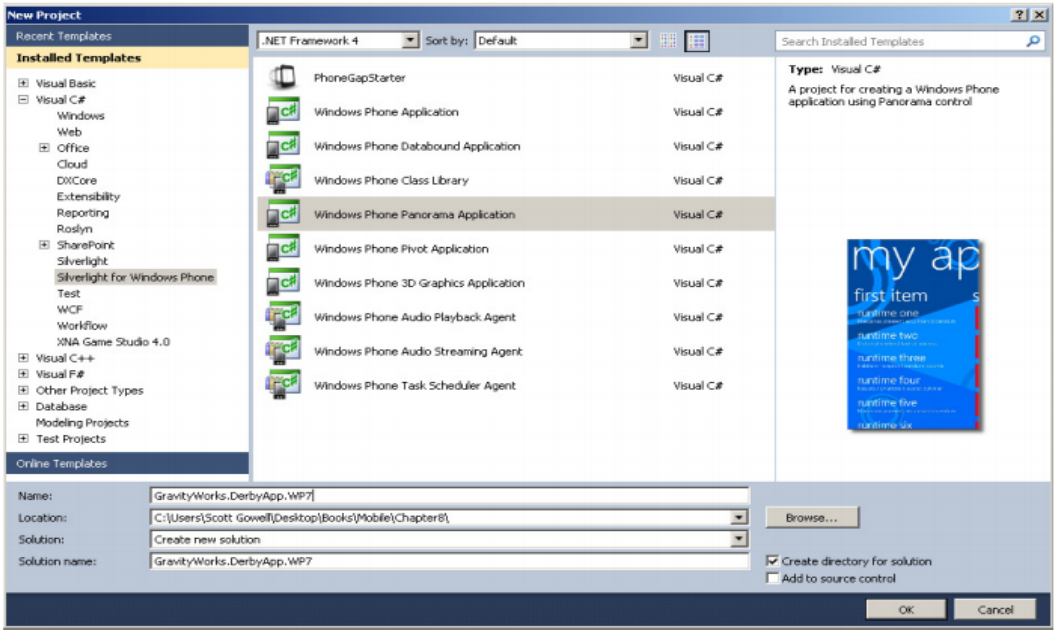
Creating the Project:

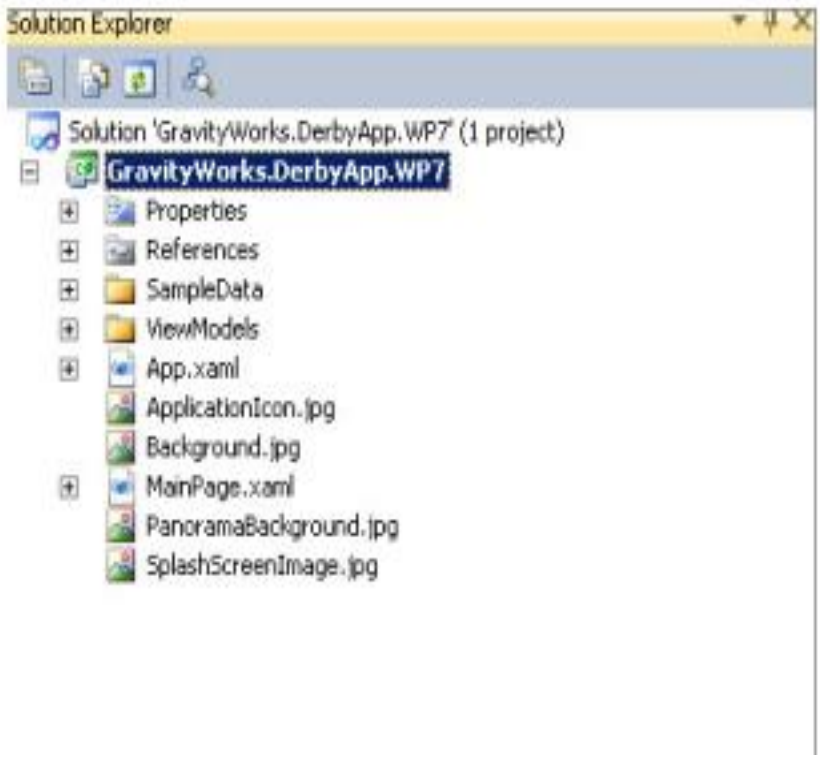
Open Visual Studio and create a new Windows Phone project. For this application, choose Panorama because it offers a UI in which you can share your data.

In a Panorama application the application is created with the default Panorama background.

Visual Studio will create SampleData and ViewModels for your application. Ultimately, you will be able to remove these from your application when you implement your service communications.

App.xaml is the entry point for your application and MainPage.xaml is the page that loads by default.





User Interface:
The default Panorama application defines its DataContext in XAML. The DataContext has first item's binding associated by default. The Panorama control can be likened to any collection-based UI element (UITableView in iOS or the ListView in Android), and the PanoramaItems are the respective rows in that collection element. When you feel familiar enough to start working with the data you will need to create a service reference to the Odata feed.

To reference an OData feed you need only to right-click your project, choose Add Service Reference, enter in the URL of your service, and click Go. After it has found the service it should enumerate the models. You are then allowed to update the namespace and create this reference. Once you create the service you can start working with the Panorama control to bind the data available from these entities. After you have made this service, be sure to reference this entity context when your page needs to make calls to the service:
readonly DerbyNamesEntities context = new DerbyNamesEntities(new Uri("http://localhost:1132/DerbyNames.svc/"));

Derby Names:
To bind data to your Panorama item you need to set the ItemsSource and TextBlock bindings. Each individual entry in the DerbyNames entity in OData contains properties for Name and League, which you will bind to the TextBlocks in your Panorama item.

Leagues: Each derby team belongs to a league. The entity for League is similar to the DerbyNames entity, and will make it easy to bind from.

9.Implement an application that creates an alert upon receiving a message.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_margin="10dp"
   android:orientation="vertical">

   <TextView
      android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="Message"
    android:textSize="30sp" />

<EditText
    android:id="@+id/editText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:textSize="30sp" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="30dp"
    android:layout_gravity="center"
    android:text="Notify"
    android:textSize="30sp"/>

  </LinearLayout>
```

Code for MainActivity.java:

```
package com.example.lab9;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {

        Button notify;
         EditText e;

  @Override
  protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    notify= (Button) findViewById(R.id.button);
    e= (EditText) findViewById(R.id.editText);

    notify.setOnClickListener(new View.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        Intent intent = new Intent(MainActivity.this, MainActivity.class);
        PendingIntent pending = PendingIntent.getActivity(MainActivity.this, 0,
      intent, 0);
```

```
            Notification noti = new
          Notification.Builder(MainActivity.this).setContentTitle("New
          Message").setContentText(e.getText().toString()).setSmallIcon(R.drawable.i
          c_launcher).setContentIntent(pending).build();
            NotificationManager manager = (NotificationManager)
          getSystemService(NOTIFICATION_SERVICE);
            noti.flags |= Notification.FLAG_AUTO_CANCEL;
            manager.notify(0, noti);
          }
        });
    }
}
```

10.Create an application that writes data to the SD card.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_margin="20dp"
android:orientation="vertical">
<EditText
android:id="@+id/editText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:singleLine="true"
android:textSize="30dp" />
<Button
android:id="@+id/button"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="10dp"
android:text="Write Data"
android:textSize="30dp" />
<Button
android:id="@+id/button2"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="10dp"
android:text="Read data"
android:textSize="30dp" />
<Button
android:id="@+id/button3"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_margin="10dp"
android:text="Clear"
android:textSize="30dp" />
</LinearLayout>
```

Manifest for the Android Application:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-
permission>
```

MainActivity.java

```
package com.example.lab8;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity {
EditText e1;
Button write,read,clear;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
e1= (EditText) findViewById(R.id.editText);
write= (Button) findViewById(R.id.button);
read= (Button) findViewById(R.id.button2);
clear= (Button) findViewById(R.id.button3);
write.setOnClickListener(new View.OnClickListener()
{
@Override
public void onClick(View v)
{
String message=e1.getText().toString();
try
{
File f=new File("/sdcard/myfile.txt");
f.createNewFile();

FileOutputStream fout=new

FileOutputStream(f);
fout.write(message.getBytes());

fout.close();
Toast.makeText(getBaseContext(),"Data

Written in SDCARD",Toast.LENGTH_LONG).show();
}
catch (Exception e)
{

Toast.makeText(getBaseContext(),e.getMessage(),Toast.LENGTH_LO
NG).show();
}
}
});
read.setOnClickListener(new View.OnClickListener()
{
@Override
```

```
public void onClick(View v)
{
String message;
String buf = "";
try
{
File f = new File("/sdcard/myfile.txt");
FileInputStream fin = new
FileInputStream(f);
BufferedReader br = new BufferedReader(new
InputStreamReader(fin));
while ((message = br.readLine()) != null)
{
buf += message;
}
e1.setText(buf);
br.close();
fin.close();
Toast.makeText(getBaseContext(),"Data

Recived from SDCARD",Toast.LENGTH_LONG).show();
}
catch (Exception e)
{
Toast.makeText(getBaseContext(),
e.getMessage(), Toast.LENGTH_LONG).show();
}
}
});
clear.setOnClickListener(new View.OnClickListener()
{
@Override
public void onClick(View v)
{
e1.setText("");
}
});
}
}
```