

CMR Institute of Technology

Mobile Applications -18MCA53

Answer Key-December 2019

1.a. Explain the preliminary considerations to build a mobile application.

A developer may go for mobile app development because of following reasons –

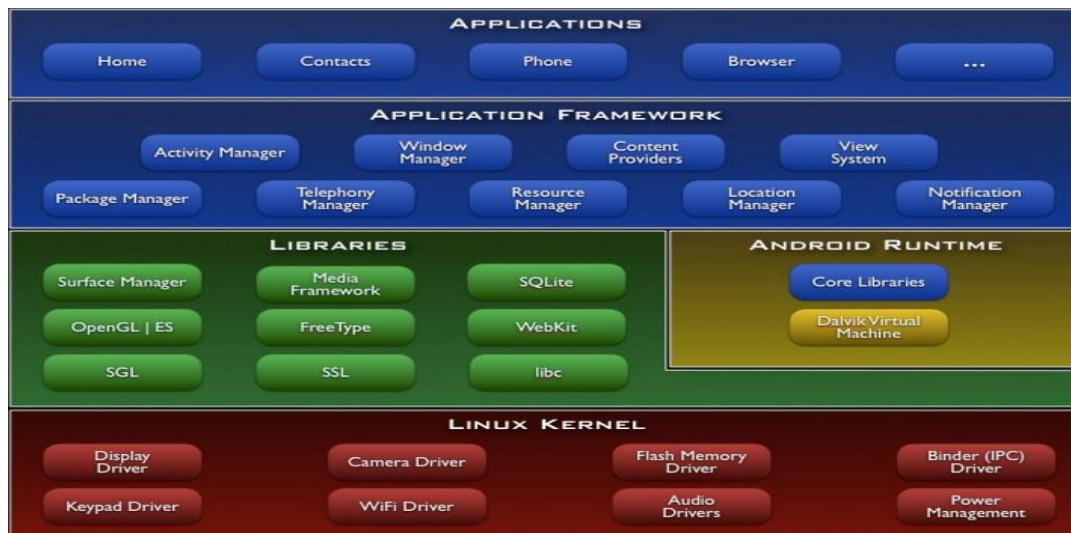
☐ Your competitors have mobile apps, but you don't. So, to improve the business and to sustain in the market, one may like to go for having a mobile-app.

☐ Mobile apps make good business sense. One can give good quality service in a faster time frame.

☐ Your services would add value to a user's mobile experience but your website isn't mobile friendly.

☐ Do you need a mobile application or a mobile website?

1.b. Outline the architecture of android.



The Android OS is roughly divided into five sections in four main layers:

Linux kernel — This is the kernel on which Android is based. This layer contains all the lowlevel device drivers for the various hardware components of an Android device.

Libraries — These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

Android runtime — At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine (Android applications are compiled into the Dalvik executables). Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

Application framework — Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.

Applications — At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

2.a Compare and contrast between mobile platforms.

1 Android

- Android has a diverse ecosystem, with fewer institutionalized restrictions and a wider variety of mobile devices than other popular systems.
- They are strong competitor in the mobile market, but the flexibility of Android design can introduce new issues.
- Development of the Android operating system is led by Google
http://developer.android.com/guide/practices/ui_guidelines/index.html

Interface Tips

– Get started on Android application design with these hints.

- Android convention is to place view-control tabs across the top, and not the bottom, of the screen.
- Use the main application icon for temporal, hierarchical navigation, instead of a “back” button and main icon link to the home screen.
- Don’t mimic user interface elements or recycle icons from other platforms.
- For instance, list items should not use carets to indicate deeper content.
- Parallax scrolling is common in Android applications.
- Android development can extend to home-screen “widget” tools.

Accessibility

- Google provides guidelines and recommendations, such as testing with the often-preinstalled and always-free Talkback. Accessibility design guidelines are listed on the Android Developer website (<http://developer.android.com/guide/topics/ui/accessibility/index.html>), and further discussed by the Google “Eyes Free” project (http://eyesfree.googlecode.com/svn/trunk/documentation/android_access/index.html).

2 iOS

- Apple maintains strict design standards, which are detailed and updated online. iOS interface documentation and general mobile design strategies are available from Apple, including design strategies and case studies, at <http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html>

Interface Tips

- -Apple can reject an application from the official App Store because of design problems. Follow the current guidelines closely, starting with these tips:
- ☒ iPhone users generally hold from the bottom of the device, so main navigation items should be in reach of user thumbs.
- ☒ Target areas for controls should be a minimum of 44 x 44 points.
- ☒ Support standard iOS gestures, such as swiping down from the top to reveal the Notification Center.
- ☒ Larger iPad screens are great for custom multi-finger gestures, but nonstandard gestures should never be the only way to reach and use important features.

Accessibility

- ☒ Accessible touch and gestural controls are available on the iPad and later generation iPhones;
- ☒ Screen magnification and colour contrast adjustments are also available.






3 BlackBerry OS

- BlackBerry OS is often the mobile device of choice in or corporate environments.

- BlackBerry includes native support of corporate emails; and runs on many devices with hard keypads. Which is favoured by users with accessibility issues as well as late adopters to touch-screen interfaces.

Interface Tips

- Link common tasks to the BlackBerry track pad according to standard actions:

-  Press the track pad: Default option, like revealing the menu
-  Press and hold track pad: Activate available pop-up box
-  Press track pad and select Shift: Highlight content
-  Press track pad and select Alt: Zoom
-  Move finger along track pad: Cursor or mouse will move accordingly


Accessibility

- BlackBerry mobile devices include text-based push-delivery messages, closed captions on multimedia content, and hearing-aid compatibility for hearing accessibility issues. Low-vision users can use the Clarity theme and other screen adjustments, and benefit from tactile keyboards. Predictive text and AutoText aid users with mobility and cognitive issues. Best practices and device capabilities are maintained online at <http://docs.blackberry.com/en/>

4 Windows Phone7

- Developed by Microsoft, Windows Phone 7 (WP7) is a currently smaller contender, focused on consumer markets.
- Using the “Metro” theme, features are divided into “Live Tiles” that link to applications.
- Six dedicated hardware buttons (back, start, search, camera, power, and volume), at least 4 GB of Flash memory, and Assisted GPS.

Interface Tips

- Windows Phone 7 interfaces are minimalist,
-  Using empty space to lend clarity to the application. WP7 uses movement over gradients for on-screen elements to immerse users in the application experience. Users will enter a WP7 application from a “tile,” which can display dynamic and real-time information.

- ☒ Tile images should be in the PNG format, 173 pixels 173 pixels at 256 dpi. leaving focus. Do not use a “back” button to navigate back the page stack. Panorama controls slide horizontally through panes, and pivot controls list panes users can visit.
- Uniform Page Shuffle presents non-hierarchical information users can shuffle through;
 - ☒ “leaf-blowing turn” flips content area into focus,
 - ☒ Scattering and tilting tiles leaving focus.

Accessibility

- WP7 devices include many standard accessibility features, such as color and
- Contrast adjustment to themes for low-vision users. Many, but not all, devices
- are compatible with TTY, TDD, and hearing aids.
- Learn more about the basics of WP7 accessibility at
- <http://www.microsoft.com/>

5 Mobile Web Browsers

- If a mobile application sends users to a website, that website should be optimized for mobile browsers.
- Similarly, mobile web applications should follow key mobile design methods.
- A great resource for design best practices for mobile web browsers is published by the W3C.

Interface Tips

Few quick tips to get started:

- ☒ Test for a consistent experience: when websites are accessed from a variety of mobile browsers.
- ☒ Provide minimal navigation: at the top of the page, and use consistent navigation mechanisms.
- ☒ Do not change or refresh the current window: or cause pop-ups, without informing the user and providing the means to stop it.
- ☒ Limit content: what the user has requested, and what the user’s device can display by avoiding large image files.
- ☒ Specify default input formats : when possible, provide preselected defaults

Accessibility

- The W3C Web Accessibility Initiative provides introductions, solutions, and further resources to create accessible mobile websites and mobile web applications

2.b Explain the terms to understand mobile application users

The **Gestalt principles** have had a considerable influence on design, describing how the human mind perceives and organizes visual data. The Gestalt principles refer to theories of visual perception developed by German psychologists in the 1920s. According to these principles, every cognitive stimulus is perceived by users in its simplest form. Key principles include **proximity, closure, continuity, figure and ground, and similarity**.

Proximity:

- Users tend to group objects together.
- Elements placed near each other are perceived in groups as shown in Figure 1.1.



Figure 1.1 Proximity

- Many smaller parts can form a unified whole.
- Icons that accomplish similar tasks may be categorically organized with proximity.
- Place descriptive text next to graphics so that the user can understand the relationship between these graphical and textual objects.

Closure:

- If enough of a shape is available, the missing pieces are completed by the human mind.
- In perceiving the unenclosed spaces, users complete a pattern by filling in missing information. For example, people recognize it as a triangle even though the Figure 1.2 is not complete.
- In grid patterns with horizontal and vertical visual lines, use closure to precisely show the inside and outside of list items.



Figure 1.2 Closure

Continuity:

- The user's eye will follow a continuously-perceived object. When continuity occurs, users are compelled to follow one object to another because their focus will travel in the direction they are already looking.
- They perceive the horizontal stroke as distinct from the curled stroke in the Figure 1.3, even though these separate elements overlap.



Figure 1.3 Continuity

- Smooth visual transitions can lead users through a mobile application, such as a link with an indicator pointing toward the next object and task.

Figure and Ground:

- A figure, such as a letter on a page, is surrounded by white space, or the ground. For example, in Figure 1.4, the figure is the gear icon, and the ground is the surrounding space.



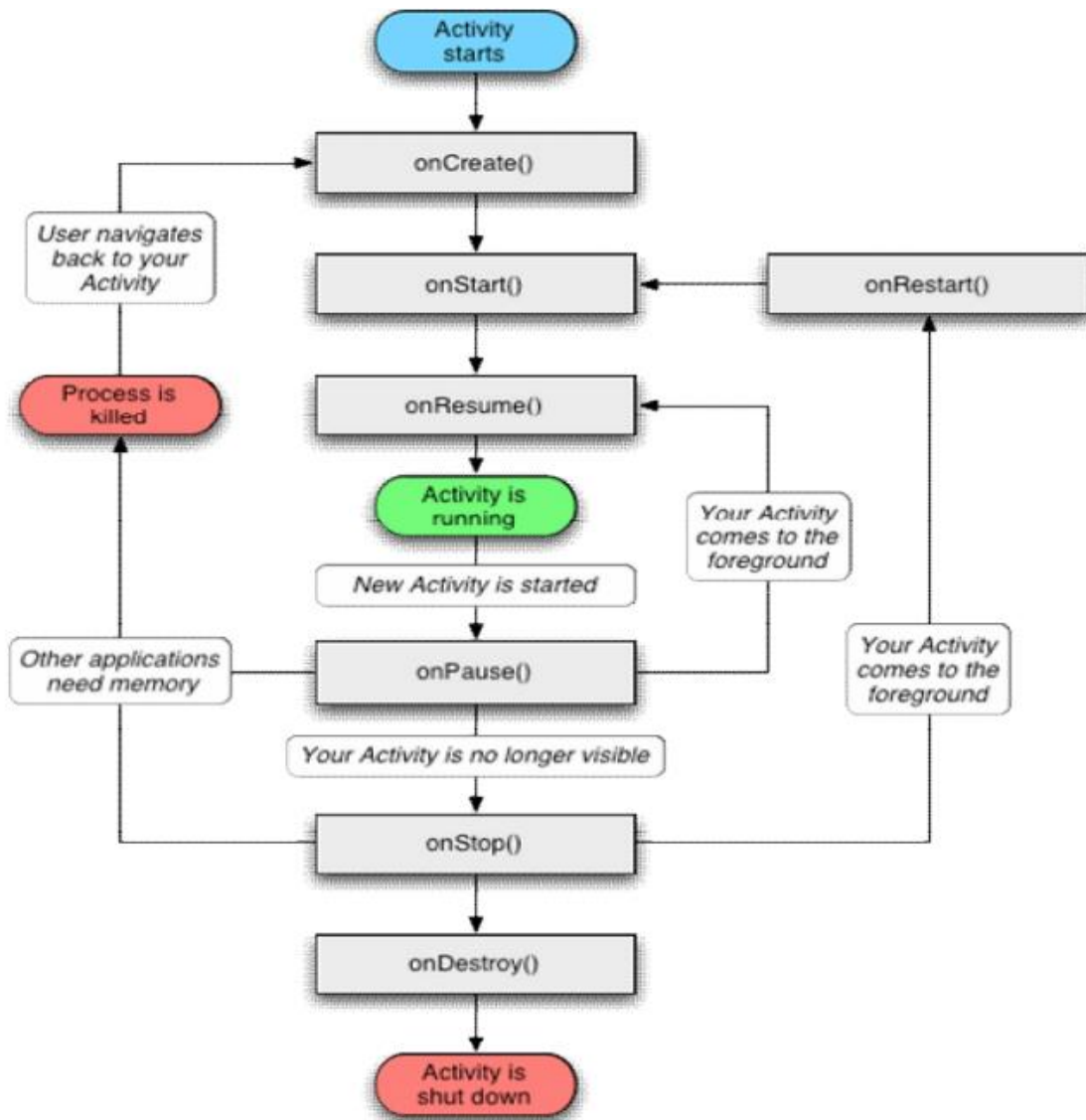
Figure 1.4 Figure and Ground

- Primary controls and main application content should maintain a distinct separation between figure and ground.

Similarity:

- Similar elements are grouped in a semi-automated manner, according to the strong visual perception of colour, form, size, and other attributes. Figure 1.5 illustrates it.
- In perceiving similarity, dissimilar objects become emphasized.
- Strict visual grids confuse users by linking unrelated items within the viewport.
- The layout should encourage the proper grouping of objects and ideas.

3.a. Explain briefly the life cycle of an activity and fragments.



The Activity class defines the following events:

- ☐ `onCreate()` — Called when the activity is first created
- ☐ `onStart()` — Called when the activity becomes visible to the user
- ☐ `onResume()` — Called when the activity starts interacting with the user
- ☐ `onPause()` — Called when the current activity is being paused and the previous activity is being resumed
- ☐ `onStop()` — Called when the activity is no longer visible to the user
- ☐ `onDestroy()` — Called before the activity is destroyed by the system (either manually

Activity's process is terminated without completing its full lifecycle. The `onAttach` event is triggered before the Fragment's UI has been created, before the Fragment itself or its parent Activity have finished their initialization. Typically, the `onAttach` event is used to gain a reference to the parent Activity in preparation for further initialization tasks.

📌 **Creating and Destroying Fragments:** The created lifetime of your Fragment occurs between the first call to `onCreate` and the final call to `onDestroy`. As it's not uncommon for an Activity's process to be terminated without the corresponding `onDestroy` method being called, so a Fragment can't rely on its `onDestroy` handler being triggered. As with Activities, you should use the `onCreate` method to initialize your Fragment. It's good practice to create any class scoped objects here to ensure they're created only once in the Fragment's lifetime.

📌 **Creating and Destroying User Interfaces:** A Fragment's UI is initialized (and destroyed) within a new set of event handlers: `onCreateView` and `onDestroyView`, respectively. Use the `onCreateView` method to initialize your Fragment:

- o Inflate the UI,
- o get references (and bind data to) the Views it contains,
- o and then create any required Services and Timers.

Once you have inflated your View hierarchy, it should be returned from this handler:

```
return inflater.inflate(R.layout.my_fragment, container, false);
```

3.b Describe anatomy of an android application.

The various folders and their files are as follows:

📌 **src** — Contains the file, ***MainActivity.java***. It is the source file for your activity. You will write the code for your application in this file.

☞ **Android 4.4.2** — This item contains one file, **android.jar**, which contains all the class libraries needed for an Android application.

☞ **gen** — Contains the **R.java** file, a compiler-generated file that references all the resources found in your project. **You should not modify this file.**

☞ **assets** — This folder contains all the assets used by your application, such as HTML, text files, databases, etc.

☞ **res** — This folder contains all the resources used in your application. It also contains a few other subfolders:

- o **drawable - <resolution>**: All the image files to be used by the Android application must be stored here.

- o **layout** - contains **activity_main.xml** file, which is the GUI of the application.

- o **values** - contains files like **strings.xml**, **styles.xml** that are needed for storing the string variables used in the applications, creating style-sheets etc.

☞ **AndroidManifest.xml** — This is the manifest file for your Android application. Here you specify the permissions needed by your application, as well as other features (such as intent-filters, receivers, etc.).

Details of some of the important files are given hereunder:

☞ **strings.xml File:** The activity_main.xml file defines the user interface for your activity. Observe the following in bold:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

The **@string** in this case refers to the strings.xml file located in the res/values folder.

Hence, **@string/hello** refers to the hello string defined in the **strings.xml** file, which is "Hello World!":

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello World!</string>
<string name="app_name">HelloWorld</string>
</resources>
```

It is recommended that you store all the string constants in your application in this **strings.xml** file and reference these strings using the **@string** identifier. That way, if you ever need to localize your application to another language, all you need to do is replace the strings stored in the **strings.xml** file with the targeted language and recompile your application.

📖 **AndroidManifest.xml File:** This file contains detailed information about the application. Observe the code in this file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.HelloWorld"
android:versionCode="1"
android:versionName="1.0" >
<uses-sdk
android:minSdkVersion="19"
android:targetSdkVersion="19" />
<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
```

```
<activity
  android:name=".MainActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
</application>
</manifest>
```

Key points about this file are as below :

- o It defines the package name of the application as

net.learn2develop.HelloWorld.

- o The version code of the application is 1. This value is used to identify the version number of your application. It can be used to programmatically determine whether an application needs to be upgraded.

- o The version name of the application is 1.0. This string value is mainly used for display to the user.

- o The application uses the image named ***ic_launcher.png*** located in the ***drawable*** folder.

- o The name of this application is the string named ***app_name*** defined in the ***strings.xml*** file.

- o There is one activity in the application represented by the ***MainActivity.java*** file. The label displayed for this activity is the same as the application name.

- o Within the definition for this activity, there is an element named `<intent-filter>`:

☒ The action for the intent filter is named ***android.intent.action.MAIN*** to indicate that this activity serves as the entry point for the application.

☒ The category for the intent-filter is named ***android.intent.category.LAUNCHER*** to indicate that the application can be launched from the device's Launcher icon.

o Finally, the ***android:minSdkVersion*** attribute of the <uses-sdk> element specifies the minimum version of the OS on which the application will run.

☒ **R.java File:** As you add more files and folders to your project, Eclipse will automatically generate the content of **R.java**, which at the moment contains the following:

```
package net.learn2develop.HelloWorld;

public final class R {

    public static final class attr {

    }

    public static final class drawable {

        public static final int icon=0x7f020000;

    }

    public static final class layout {

        public static final int main=0x7f030000;

    }

    public static final class string {

        public static final int app_name=0x7f040001;

        public static final int hello=0x7f040000;

    }

}
```

You are not supposed to modify the content of the R.java file; Eclipse automatically generates the content for you when you modify your project.

🔗 **MainActivity.java File:** The code that connects the activity to the UI

(activity_main.xml) is the setContentView() method, which is in the MainActivity.java file:

```
package net.learn2develop.HelloWorld;

import android.app.Activity;

import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */

    @Override

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
    }
}
```

Here, **R.layout.main** refers to the **activity_main.xml** file located in the **res/layout** folder. As you add additional XML files to the **res/layout** folder, the filenames will automatically be generated in the **R.java** file. The **onCreate()** method is one of many methods that are fired when an activity is loaded.

4.a. Write short notes on components of android application.

There are four different types of app components:

1. Activities

2. Services
3. Broadcast receivers
4. Content providers

Activities

An activity is the entry point for interacting with the user. It represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities if the email app allows it. For example, a camera app can start the activity in the email app that composes new mail to allow the user to share a picture. An activity facilitates the following key interactions between system and app:

Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.

Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.

Helping the app handle having its process killed so the user can return to activities with their previous state restored. Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

You implement an activity as a subclass of the Activity class. For more information about the Activity class, see the Activities developer guide.

Services

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are actually two very distinct semantics services tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represent two different types of started services that modify how the system handles them:

Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.

A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about. Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running.

A service is implemented as a subclass of `Service`. For more information about the `Service` class, see the [Services developer guide](#).

Broadcast receivers

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event... and by delivering that alarm to a `BroadcastReceiver` of the app, there is no need for the app to remain running until the alarm goes off. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. For

instance, it might schedule a JobService to perform some work based on the event with JobScheduler

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see the BroadcastReceiver class.

Content providers

A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it. For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query the content provider, such as ContactsContract.Data, to read and write information about a particular person

4.b. Differentiate between linear and absolute layout.

LinearLayout

☑ The LinearLayout arranges views in a single column or a single row.

☑ Child views can be arranged either vertically or horizontally.

☑ To see how LinearLayout works, consider the following elements typically contained

in the activity_main.xml file:

```
<?xml version="1.0" encoding="Utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="Vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="string/hello" />
```

```
</LinearLayout>
```

☒ In the main.xml file, observe that the root element is <LinearLayout> and

☒ It has a <TextView> element contained within it.

☒ The <LinearLayout> element controls the order in which the views contained within it appear.

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. Consider the following UI defined in main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android=http://schemas.android.com/apk/res/android>
    <Button
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_x="126px"
        android:layout_y="361px"/>
    <Button
        android:layout_width="113dp"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_x="12px"
        android:layout_y="361px"/>
```

</AbsoluteLayout>

5.a. Describe views and view groups in android application with syntax.

Some of the basic views that can be used to design UI of Android application are:

- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup

These basic views enable you to display text information, as well as perform some basic selection.

☐ **TextView View:** The TextView view is used to display text to the user. When we create a new Android project, Eclipse always creates one <TextView> element in activity_main.xml file, to display Hello World as shown below –

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>
```

Button — Represents a push-button widget

ImageButton — Similar to the Button view, but it also displays an image

EditText — A subclass of the TextView view, but it allows users to edit its text content

CheckBox — A special type of button that has two states: checked or unchecked

RadioGroup and **RadioButton** — The RadioButton has two states: either checked or unchecked. Once a RadioButton is checked, it cannot be unchecked. A RadioGroup is used to group together one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.

ToggleButton — Displays checked/unchecked states using a light indicator

To understand the behavior of these views, create a new android project and place the following code in activity_main.xml file, without disturbing existing code.

```
<Button android:id="@+id/btnSave"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Save" />

<Button android:id="@+id/btnOpen"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Open" />

<ImageButton android:id="@+id/btnImg1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:src="@drawable/icon" />

<EditText android:id="@+id/txtName"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />

<CheckBox android:id="@+id/chkAutosave"
android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content"
android:text="Autosave" />
<CheckBox android:id="@+id/star"
style="?android:attr/starStyle"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
<RadioGroup android:id="@+id/rdbGp1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:orientation="vertical" >
<RadioButton android:id="@+id/rdb1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Option 1" />
<RadioButton android:id="@+id/rdb2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Option 2" />
</RadioGroup>
<ToggleButton android:id="@+id/toggle1"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

After running the application, the output will be displayed as shown in the following diagram. Click on each of these views, to observe their default behavior.

5.b Write short notes on views and spinner views

List views are views that enable you to display a long list of items. In Android, there are two types of list views: ListView and SpinnerView. Both are useful for displaying long lists of items. The ListView displays a list of items in a vertically scrolling list.

To add SpinnerView in the android application, use the following code:

```
<Spinner android:id="@+id/spinner1"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:drawSelectorOnTop="true" />
```

6.a. Outline the working of Google maps in android applications.

Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things. This section describes how to use Google Maps in your Android applications and programmatically perform the following:

- ☐ Change the views of Google Maps.
- ☐ Obtain the latitude and longitude of locations in Google Maps.
- ☐ Perform geocoding and reverse geocoding (translating an address to latitude and longitude and vice versa).
- ☐ Add markers to Google Maps.

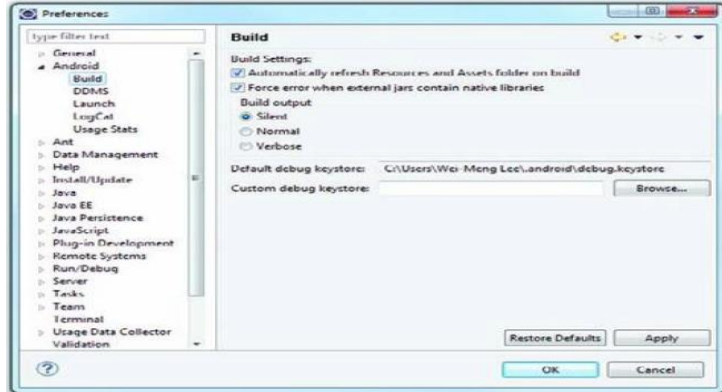
We will discuss how to build a project using maps.

Creating the Project: Create a new android project. In order to use Google Maps in your Android application, you need to ensure that you check the Google APIs as your build target. Google Maps is not part of the standard Android SDK, so you need to find it in the Google APIs add-on. If LBS is the name of your project, then you can see the additional JAR file (maps.jar) located under the Google APIs folder as below–



Obtaining the Maps API Key: Beginning with the Android SDK release v1.0, you need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application. When you apply for the key, you must also agree to Google's terms of

use, so be sure to read them carefully.



First, if you are testing the application on the Android Emulator or an Android device directly connected to your development machine, locate the SDK debug certificate located in the default folder (C:\Users\\.android for Windows 7 users). You can verify the existence of the debug certificate by going to Eclipse and selecting Window ⇄ Preferences. Expand the Android item and select Build (as shown in figure above). On the right side of the window, you will be able to see the debug certificate's location. The filename of the debug keystore is debug.keystore. This is the certificate that Eclipse uses to sign your application so that it may be run on the Android Emulator or devices.

6.b Illustrate the three methods of getting location based data.

Nowadays, mobile devices are commonly equipped with **GPS receivers**.

- o Because of the many satellites orbiting the earth, you can use a GPS receiver to find your location easily. However, GPS requires a clear sky to work and hence does not always work indoors or where satellites can't penetrate (such as a tunnel through a mountain).

- o Another effective way to locate your position is through **cell tower triangulation**.

- o When a mobile phone is switched on, it is constantly in contact with base stations surrounding it.

- o By knowing the identity of cell towers, it is possible to translate this information into a physical location through the use of various databases

containing the cell towers' identities and their exact geographical locations.

- o The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites.

- o It is not as precise as GPS because its accuracy depends on overlapping signal coverage, which varies quite a bit.

- o Cell tower triangulation works best in densely populated areas where the cell towers are closely located.

☒ A third method of locating your position is to rely on **Wi-Fi triangulation**.

- o Rather than connect to cell towers, the device connects to a Wi-Fi network and checks the service provider against databases to determine the location serviced by the provider

On the Android, the SDK provides the LocationManager class to help your device determine the user's physical location.

```
lm.requestLocationUpdates( LocationManager.GPS_PROVIDER, 0, 0,  
locationListener);
```

This method takes four parameters:

1. Provider -The name of the provider with which you register. In this case, you are using GPS to obtain your geographical location data.
2. minTime - The minimum time interval for notifications, in milliseconds.
3. minDistance - The minimum distance interval for notifications, in meters.
4. Listener - An object whose onLocationChanged() method will be called for each location update

7.a. Explain briefly how sms and mail works in android.

SMS messaging is one of the main *killer applications* on a mobile phone today — for some users as necessary as the phone itself. Any mobile phone you buy today should have at

least SMS messaging capabilities, and nearly all users of any age know how to send and receive such messages. Android comes with a built-in SMS application that enables you to send and receive SMS messages. However, in some cases you might want to integrate SMS capabilities into your own android application. For example, you might want to write an application that automatically sends a SMS message at regular time intervals. For example, this would be useful if you wanted to track the location of your kids — simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes.

4.1.1 Sending SMS Messages Programmatically

To create an application that can send SMS, following are the steps to be followed:

❑ Create a new android application.

❑ Add the following statements in to the main.xml file:

```
<Button  
    android:id="@+id/btnSendSMS"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Send SMS" />
```

❑ In the AndroidManifest.xml file, add the following statements:

```
<uses-sdk android:minSdkVersion="8" />  
  
<uses-permission  
    android:name="android.permission.SEND_SMS">  
  
</uses-permission>
```

❑ Add the following statements to the MainActivity.java file:

```
import android.app.PendingIntent;  
  
import android.content.Intent;
```

```
import android.telephony.SmsManager;

import android.view.View;

import android.widget.Button;

public class MainActivity extends Activity

{

    Button btnSendSMS;

    /** Called when the activity is first created.
    @Override

    public void onCreate(Bundle savedInstanceState)

    {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        btnSendSMS = (Button) findViewById(R.id.btnSendSMS);

        btnSendSMS.setOnClickListener(new View.OnClickListener()

        {

            public void onClick(View v)

            {

                sendSMS("5556", "Hello my friends!");

            }

        });

    }

    private void sendSMS(String phoneNumber, String message)

    {

        SmsManager sms = SmsManager.getDefault();

        sms.sendTextMessage(phoneNumber, null, message, null, null);

    }

}
```

```
}
```

Following are the five arguments to the `sendTextMessage()` method:

- ☐ `destinationAddress` — Phone number of the recipient
- ☐ `scAddress` — Service center address; use null for default SMSC
- ☐ `text` — Content of the SMS message
- ☐ `sentIntent` — Pending intent to invoke when the message is sent
- ☐ `deliveryIntent` — Pending intent to invoke when the message has been

Delivered

One can set email through Android program. Following are the steps involved:

- ☐ Create a new android application.
- ☐ Add the following statements in to the main.xml file:

```
<Button  
    android:id="@+id/btnSendEmail"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Send Email" />
```

- ☐ Add the following statements in bold to the MainActivity.java file:

```
import android.content.Intent;  
import android.net.Uri;  
import android.view.View;  
import android.widget.Button;  
  
public class MainActivity extends Activity  
{  
    Button btnSendEmail;  
  
    /** Called when the activity is first created. */  
  
    @Override
```

```

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnSendEmail = (Button) findViewById(R.id.btnSendEmail);
    btnSendEmail.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View v)
        {
            String[] to = {"weimenglee@learn2develop.net",
                "weimenglee@gmail.com"};
            String[] cc = {"course@learn2develop.net"};
            sendEmail(to,cc,"Hello", "Hello my friends!");
        }
    });
}

//---sends an SMS message to another device---
private void sendEmail(String[] emailAddresses, String[]
    carbonCopies, String subject, String message)
{
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    String[] to = emailAddresses;
    String[] cc = carbonCopies;
    emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
}

```

```
emailIntent.putExtra(Intent.EXTRA_CC, cc);
emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, message);
emailIntent.setType("message/rfc822");
startActivity(Intent.createChooser(emailIntent, "Email"));
}
}
```

7.b How do you publish android applications? List out the steps briefly.

Google has made it relatively easy to publish your Android application so that it can be quickly distributed to end users. The steps to publishing your Android application generally involve the following:

- ☐ Export your application as an APK (Android Package) file.
- ☐ Generate your own self-signed certificate and digitally sign your application with it.
- ☐ Deploy the signed application.
- ☐ Use the Android Market for hosting and selling your application.

Here, you will learn how to prepare your application for signing, and then learn about the various ways to deploy your applications.

Versioning: Beginning with version 1.0 of the Android SDK, the AndroidManifest.xml file of every Android application includes the android:versionCode and android:versionName attributes:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="net.learn2develop.LBS"
android:versionCode="1"
android:versionName="1.0">
```

The `android:versionCode` attribute represents the version number of your application. For every revision you make to the application, you should increment this value by 1 so that you can programmatically differentiate the newest version from the previous one. This value is never used by the Android system, but is useful for developers as a means to obtain the version number of an application. However, the `android:versionCode` attribute is used by Android Market to determine if there is a newer version of your application available.

The `android:versionName` attribute contains versioning information that is visible to the users. If you are planning to publish your application on the Android Market (means, playstore), the `AndroidManifest.xml` file must have the following attributes:

- o `android:versionCode` (within the `<manifest>` element)
- o `android:versionName` (within the `<manifest>` element)
- o `android:icon` (within the `<application>` element)
- o `android:label` (within the `<application>` element)

The `android:label` attribute specifies the name of your application. This name will be displayed in the Settings \rightarrow Applications \rightarrow Manage Applications section of your Android device.

In addition, if your application needs a minimum version of the SDK, you can specify it in the `AndroidManifest.xml` file using the `<uses-sdk>` element:

```
<uses-sdk android:minSdkVersion="7" />
```

Digitally Signing your Android Application: All Android applications must be digitally signed before they are allowed to be deployed onto a device (or emulator).

Unlike some mobile platforms, you need not purchase digital certificates from a certificate authority (CA) to sign your applications. Instead, you can generate your

own self-signed certificate and use it to sign your Android applications.

When you use Eclipse to develop your Android application and then press F11 to deploy it to an emulator, Eclipse automatically signs it for you. You can verify this by going to Windows ⇨ Preferences in Eclipse, expanding the Android item, and selecting Build. Eclipse uses a default debug keystore (appropriately named “debug.keystore”) to sign your application. A keystore is commonly known as a digital certificate.

If you are publishing an Android application, you must sign it with your own certificate. Applications signed with the debug certificate cannot be published. While you can manually generate your own certificates using the keytool.exe utility provided by the Java SDK, Eclipse has made it easy for you by including a wizard that walks you through the steps to generate a certificate. It will also sign your application with the generated certificate (which you can also sign manually using the jarsigner.exe tool from the Java SDK).

8.a. Analyse the methods for accessing web services through android application.

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.

8.b What is content provider in android? List and explain different content providers.

In Android, using a content provider is the recommended way to share data across packages.

Think of a content provider as a data store. How it stores its data is not relevant to the application using it; what is important is how packages can access the data stored in it using a consistent programming interface. A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete its content. However, unlike a database, a content provider can use different ways to store its data. The data can be stored in a database, in files, or even over a network.

Android ships with many useful content providers, including the following:

- Browser — Stores data such as browser bookmarks, browser history, and so on
- CallLog — Stores data such as missed calls, call details, and so on
- Contacts — Stores contact details
- MediaStore — Stores media files such as audio, video and images
- Settings — Stores the device's settings and preferences

9.a. Explain briefly the steps to create native ios application.

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

5.2.1 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.

5.2.2 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.

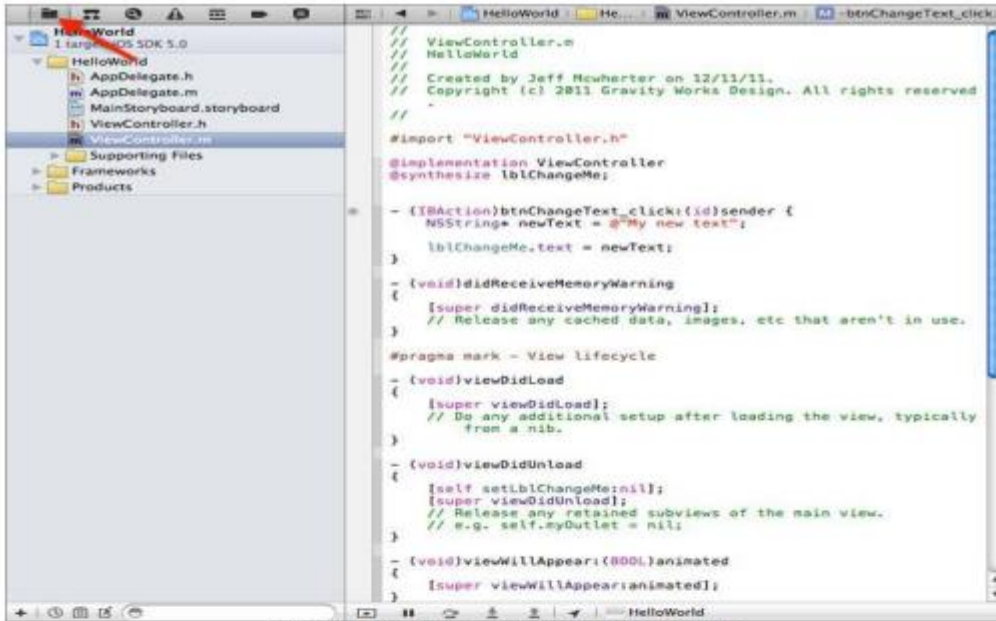


Figure 5.1 Project Navigator window

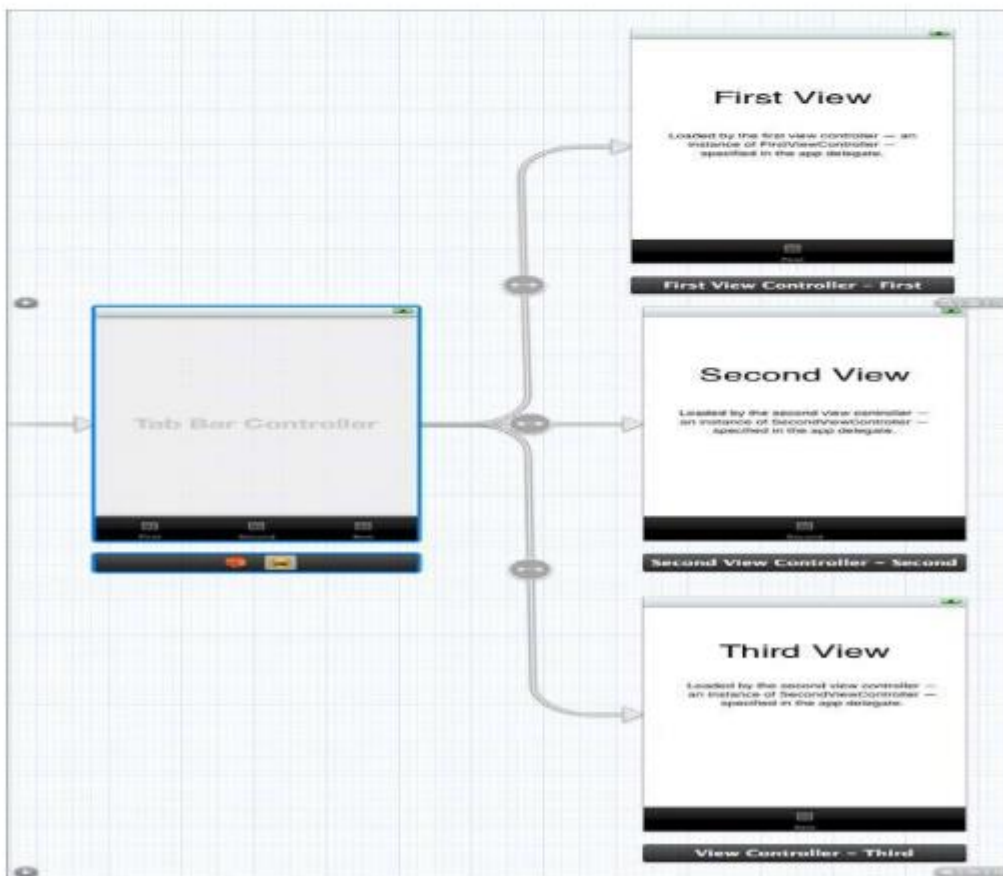


Figure 5.2 Storyboard

9.b. Give the hardware, tools and installations required to set up windows phone SDK developing software for windows phone.

To develop software for Windows Phone 7 you need a machine running Windows (Vista or Windows 7), Visual Studio 2010, and the Phone SDK, as well as a Windows Phone 7 device to test with.

Hardware: HTC, Nokia, and Samsung are currently manufacturing Windows Phone 7 devices.

Device resolutions are 800X480 pixels, and most are outfitted with both front- and rear-facing cameras, and screens from 4.3 to 4.7 inches. They are primarily found on GSM carriers.

Visual Studio and Windows Phone SDK:

☐ Code for the Windows Phone is written in the .NET Framework, either with XNA (Microsoft's run time for game development), or a custom version of Silverlight (Microsoft's Rich Internet Application framework).

- User interfaces are created with XAML (eXtensible Application Markup Language).
- The Windows Phone SDK works with Visual Studio Express, and will install it if you don't already have it installed.

– If you have another version of Visual Studio 2010 on your machine it will add the functionality to that install.

– The SDK also installs a specialized version of Expression Blend set up to work specifically for Windows Phone 7 development.

- Expression Blend is a tool developed by Microsoft for working with XAML.

– It provides similar functionality to Visual Studio, though its layout is designed to be more user friendly to designers.

Installation:

- Microsoft's App Hub (<http://create.msdn.com/>) is the download site for the Windows Phone SDK.

- The Windows Phone SDK installer includes:

– Visual Studio 2010 Express for Windows Phone (if you do not have another version

of Visual Studio 2010 installed)

- Windows Phone Emulator
- Windows Phone SDK Assemblies
- Silverlight 4 SDK
- Phone SDK 7.1 Extensions for XNA Game Studio
- Expression Blend for Windows Phone 7
- WCF Data Services Client for Windows Phone
- Microsoft Advertising SDK

To install the Windows Phone SDK you need:

- Vista (x86 or x64) or Windows 7 (x86 or x64)
- 4 GB of free disk space
- 3 GB of RAM
- DirectX 10 or above capable graphics card with a WDDM 1.1 driver

Getting to Know Visual Studio: Visual Studio is the integrated development environment from Microsoft for the .NET Framework. Visual Studio Express for Windows phones is a trimmed-down version of Microsoft's full retail products. It provides developers with everything they need to develop Windows Phone 7 apps. Though the interface may seem daunting to the uninitiated, it has a relatively simple learning curve. You are afforded both a WYSIWYG (What You See Is What You Get) and text-based editor.

Getting to Know Expression Blend: Expression Blend is a user interface design tool developed by Microsoft, with emphasis on a WYSIWYG design for XAML-based projects (Silverlight and WPF). Figure 5.7 shows the Blend UI with the standard tools displayed in a Windows Phone 7 Pivot application.

10.a. Write short note on ios story boards.

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.

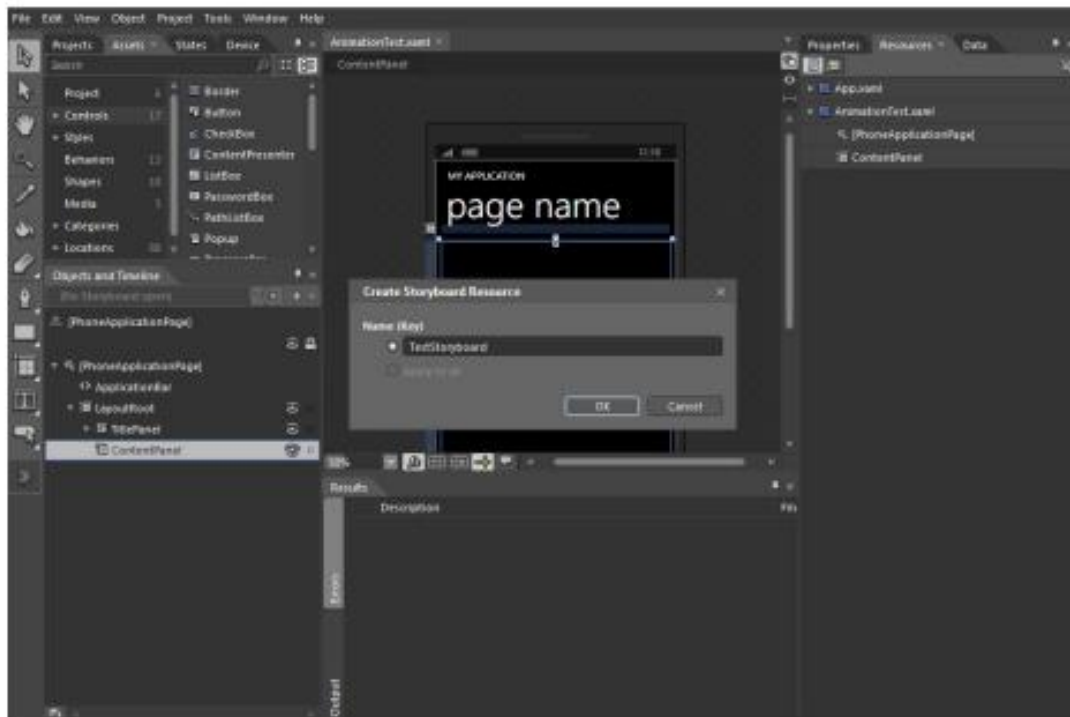


FIGURE 8-9: Storyboards in Blend

10b. Explain briefly accelerometer in windows phone7

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.