CBCS SCHEME

USN 1 CR19MCA06 18MCA52

Fifth Semester MCA Degree Examination, Feb./Mar.2022
Mobile Applications

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

1 a. Discuss preliminary cost involved in mobile application development. (10 Marks)
b. Why mobile development is difficult? (05 Marks)
c. What are the myths associated with mobile application development? (05 Marks)

OR

2 a. What are the key issues involved in using the screen real-estate effectively? (10 Marks)
b. What do you understand by mobile information design and mobile platform? (10 Marks)

Module-2

3 a. Explain the features of android. (10 Marks)
b. Explain the various layers in Android OS. (10 Marks)

OR

4 a. Briefly discuss the anatomy of an Android application. (10 Marks)
b. Discuss lifecycle of an activity with an example code. (10 Marks)

Module-3

5 a. Define a view. Explain the different types of views. (10 Marks)
b. Explain various view groups with suitable code segments. (10 Marks)

OR

6 a. How do you display Google maps in Android application explain with code? (10 Marks)
b. Discuss APK file deployment in detail. (10 Marks)

Module-4

7 a. Explain with code procedure for sending an SMS through android application. (10 Marks)
b. Write an android application to download an image file from the web. (10 Marks)

OR

8 a. Write an android application to download a text file from the web. (10 Marks)
b. Explain with code asynchronous calls in networking in view of android application. (10 Marks)

Module-5

9 a. Discuss components of iphone SDK. (10 Marks)
b. Discuss anatomy of a iOS App. (10 Marks)

OR

10 a. Discuss anatomy of a windows phone App. (05 Marks)
b. Briefly explain various steps in developing derby App in windows phone 7. (05 Marks)
c. Write a short note on notifications in windows phone 7 app and accelerometer in windows phone 7 app. (10 Marks)

* * * * *

Q1a) Discuss the preliminary cost involved in cost of development

There are many costs associated with mobile application development.

- Each developer will need hardware and software to develop the applications on.
- The team will need devices to test the software on.
- And if you want to deploy your application to any public market, then your company will need accounts on the various markets (these often renew annually).

Hardware

- To develop good mobile apps, you'll need an Intel-based Mac. Intel versions of Mac because you can run Windows on them either virtually (using something like Parallels, or VMWare Fusion) or on the bare metal (using Apple's BootCamp).
- You'll also need multiple monitors. The emulator/simulator running in one monitor, Dev Tool (IDE) running on another, and a web browser on another with the documentation for the platform for which you are developing. Having access to all of this information at once prevents context switching for a developer, and helps maintain focus.
- The emulator and simulators are great, but not perfect, so you'll need one of each of the types of
  devices you want to develop for.

Software

Following sections present an outline for what you will need for all of the platforms.

TABLE 1-1: Software Needed for Development

TARGETED FRAMEWORK	SOFTWARE REQUIRED
Window Phone 7	Windows Phone SDK Visual Studio Express Expression Blend for Windows Phone (Windows only)
iOS	xCode 4, iOS SDK xCode 4.1, iOS SDK (on Mac OS X 107) (Mac Only)
Android	Eclipse, Android SDK
BlackBerry	Eclipse, BlackBerry Plugin, BlackBerry Simulator (only works on Windows)
Titanium	Titanium Studio, Titanium Mobile SDK + Android software + iOS software
PhoneGap	PhoneGap Plugin + iOS software (Mac only) + Android software + Windows Phone 7 software (Windows only)
Any Framework Text Editors	TextMate (Mac) Notepad++ (Windows)

Licenses and Developer Accounts

The following table contains information regarding all of the various accounts necessary to develop for each platform and costs associated with such.

PLATFORM	URL	CAVEATS
BlackBerry	http://us.blackberry.com/developers/appworld/distribution.jsp	
Titanium	https://my.appcelerator.com/auth/signup/offer/community	
Windows Dev Marketplace	http://create.msdn.com/en-US/home/membership	Can submit unlimited paid apps, can submit only 100 free apps. Cut of Market Price to Store: 30%
Apple iOS Developer	http://developer.apple.com/programs/start/standard/create.php	Can only develop ad-hoc applications on up to 100 devices. Developers who publish their applications on the App Store will receive 70% of sales revenue, and will not have to pay any distribution costs for the application.
Android Developer	https://market.android.com/publish/signup	Application developers receive 70% of the application price, with the remaining 30% distributed among carriers and payment processors.

Documentation and APIs

Following are links to the respective technologies' online documentation and APIs. This will be the
location for the latest information in the respective technology.

- MSDN Library: http://msdn.microsoft.com/en-us/library/ff402535(v=vs.92).aspx
- iOS Documentation: http://developer.apple.com/devcenter/ios/index.action
- BlackBerry Documentation: http://docs.blackberry.com/en/developers/?userType=21
- Android SDK Documentation: http://developer.android.com/reference/packages.html and http://developer.android.com/guide/index.html
- PhoneGap Documentation: http://docs.phonegap.com/
- Titanium API Documentation: http://developer.appcelerator.com/apidoc/mobile/latest

The Bottom Line

- Total cost per developer to create, maintain, and distribute mobile applications for all the platforms you can expect to pay a few thousand dollars just for the minimum infrastructure. And this is really the bare minimum for development.
- Given the opportunity to expand this more I would upgrade the laptop to a MacBook Pro, with plenty of RAM, and upgrade the hard disk drive (HDD) to a solid-state drive (SSD). By making these upgrades you will incur a higher initial cost, but the speed increase compared to the bare bones will recoup that cost, if only in peace of mind.

- It is difficult to quantify the savings from these upgrades, but developers without them are at a distinct disadvantage.

Q1 b) why mobile development is difficult

The simple answer to this question is the same that plagues application developers for Mac and Windows, web developers, and mobile developers as seen from the public eye. So-called killer apps are not defined solely by what they do or how they look, but rather by how they fulfill a need and codify it for the user.

Couple that with the more intimate nature of a mobile application (I touch this and it does what I told it to do), and the more rigid (fixed size) UI design patterns of the mobile device and you get a perfect storm of potential problems.

The good news is that with proper planning and research, you target your potential clients and start imposing your own parameters on the problem at hand, and the rest can be accounted for within that scope.

Some may scoff at the limitations when looking at the resolution offerings made by Apple iOS devices, but these strict requirements afford developers dimensions they can take for granted. In

Android development, there are eleven standard potential configurations. Not all potential resolutions are actively being developed and produced, and the Android Development site tracks the progress and adoption of standard screen resolutions by device providers. Unfortunately, this makes finding the lowest common denominator more difficult, which you can see in Figures 1-1 and 1-2.

I have called out Android specifically in the following figures as it has the largest amount of different screen sizes. Additionally, the folks at Android mine this data regularly to provide exactly this type of information to developers. They understand the difficulty of accounting for all the different sizes when creating quality applications. Figure 1-1 is a pie chart that accounts for the different resource and resolution types as perceived on the Android Market. Figure 1-2 simply enumerates all the possible resolutions and pixel densities afforded for Android.

Q 1c) What are the myths associated with mobile application development?

There are many myths associated with mobile application development. It's cheap, it's easy, it's unnecessary, you can't do it without a large team, and you shouldn't have to pay for it.

Myth #1: It is inexpensive to develop a mobile solution. As previously mentioned, mobile development is not cheap. This does not include any development time, design time, and

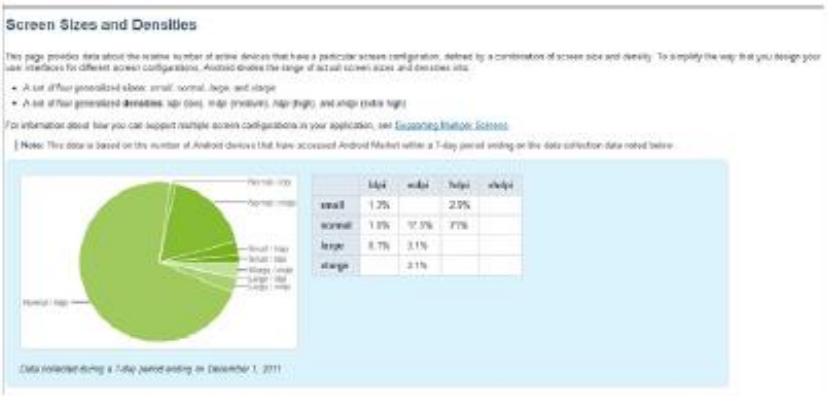
deployment time, or any potential money lost by taking too long to get to market. Iterative design and development can be expensive. Finding a happy medium is necessary to be successful when developing a mobile solution.

Myth #2: It's easy to develop a mobile solution. Future chapters discuss how to leverage existing data, use new technologies to expose that data, interpret the nuances of the native development platforms, and use the newer third-party platforms for mobile application development. In addition, later chapters attempt to make learning these topics easier than just hitting your favorite search engine and looking for tutorials. Each chapter explains each topic; this book hopefully makes the process of developing a mobile application easier. It is in no way easy.

Myth #3: We don't need a mobile presence. With the Smartphone market growing at such a large rate, and the ease with which mobile applications become available (through the market applications on the device and the markets' respective websites) there is a large set of potential customers to reach. Not everyone needs to become a mobile developer. My urge to learn mobile development came from wanting to track my newborn daughter's sleeping schedule. As new parents, my wife and I needed a solution. Two years later, I do mobile development every day, as my company's clients' needs have expanded into that market.

Myth #4: You need a large development team. Many single-developer companies are successfully releasing quality applications on the different platform markets. Certainly, a jack-of-all-trades can take an idea from wireframe to market. That being said, without a serious QA resource, development resource, and design resource it can be difficult to break away from the cookie-cutter style of applications very prevalent in the market.

Myth #5: Sweat equity can pay for the application. Not to disparage the act of creating a startup, and not to fl y in the face of innovation, but potential and dreams do not always a fortune make. Working with a partner to develop a product or solution with no capital is not easy. You've already seen the examples of what expenses to account for and resources to acquire when starting the development process. If you already have these resources, you are probably already an application developer, most likely with a 9-to-5 job or working as a contractor. There are 24 hours in the day, but they are not all billable. Eventually, something has to give; when bills come in it is generally the "side project" that falls by the wayside. Think about that before you get started. Good luck if you start on the road to becoming a contractor — it is not an easy path to travel.



HTTP://DEVELOPER.ANDROID.COM/RESOURCES/DASHBOARD/SCREENS.HTML

FIGURE 1-1: Screen sizes and densities per Google research

	Low density (120), ldpi	Medium density (160), mdpi	High density (240), hdpi	Extra high density (320), xhdpi
Small screen	QVGA (240x320)		480x640	
Normal screen	WQVGA400 (240x400) WQVGA432 (240x432)	HVGA (320x480)	WVGA800 (480x800) WVGA854 (480x854) 600x1024	640x360
Large screen	WVGA800* (480x800) WVGA854* (480x854)	WVGA800* (480x800) WVGA854* (480x854) 600x1024		
Extra Large screen	1024x600	WXGA (1280x800)† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

* To emulate this configuration, specify a custom density of 160 when creating an AVD that uses a WVGA800 or WVGA854 skin.
** To emulate this configuration, specify a custom density of 120 when creating an AVD that uses a WVGA800 or WVGA854 skin.
† This size is available with the Android 3.0 platform

HTTP://DEVELOPER.ANDROID.COM/GUIDE/PRACTICES/SCREENS_SUPPORT.HTML

FIGURE 1-2: Resolutions available to Android

Mobile development is difficult because the paradigms of design and functionality differ between it and types of development that have existed for decades. It is still new, the technologies change rapidly, and not all of the answers are known. What makes a great app different from a good app? Design? Utility? These are all things to be mindful of while developing your app.

Q2) what are the key issues involved in using the screen real estate effectively?

- The first step to use the smaller interfaces of mobile devices effectively is to know the context of use. Who are the users, what do they need and why, and how, when, and where will they access and use information?
- Mobile design is difficult, as developers try to elegantly display a telescoped view of almost limitless information. But user experience issues are amplified on mobile interfaces.
- Cognitive load increases while attention is diverted by the needs to navigate, remember what was seen, and re-find original context.
- Cognitive load is the mental effort to comprehend and use an application, whatever the inherent task complexity or information structure may be.
- Effectively use screen real estate by embracing minimalism, maintaining a clear visual hierarchy, and staying focused.

Embrace Minimalism

- Limit the features available on each screen, and use small, targeted design features.
- Content on the screen can have a secondary use within an application, but the application designer should be able to explain why that feature is taking up screen space.
- Banners, graphics, and bars should all have a purpose.

Use a Visual Hierarchy

- Help users fight cognitive distractions with a clear information hierarchy.
- Draw attention to the most important content with visual emphasis.
- Users will be drawn to larger items, more intense colors, or elements that are called out with bullets or arrows; people tend to scan more quickly through lighter color contrast, less intense shades, smaller items, and text-heavy paragraphs.
- A consistent hierarchy means consistent usability; mobile application creators can create a hierarchy with position, form, size, shape, color, and contrast.

Stay Focused

- Start with a focused strategy, and keep making decisions to stay focused throughout development.
- A smaller file size is a good indicator of how fast an application will load, so the benefits of fighting feature creep extend beyond in-application user experience.
- Focused content means users won't leave because it takes too long for the overwhelming amount of images per screen to load.
- And users won't be frustrated with the number of links that must be cycled through to complete a task. Text-heavy pages reduce engagement as eyes glaze over and users switch to another application.
- If people have taken the time to install and open an application, there is a need these users hope to meet.
- Be methodical about cutting back to user necessities. Build just enough for what users need, and knowing what users need comes from understanding users

Q2 b) What do you understand by mobile information design and mobile platform ?

Mobile devices offer an exciting space to design information, fitting personalized and real-time data into tightly-constrained screens. But keep audience goals in mind when crafting an application, because mobile devices are not generally used for extensive browsing or complex searches. Various mobile design patterns are discussed hereunder.

Information Display

People identify signals, interpret the meaning of these signals, determine the goal according to these interpretations, and then carry out an action until the goal is reached. For example, a microwave has a simple display. When the timer alerts us the popcorn is done. Overly

detailed designs do not suit mobile users, who are often micro-tasking, influenced by urgent and new surroundings, and looking for a quick fix for boredom.

Design Patterns

A design pattern recycles and repurposes components, reusing the best ideas. More than time efficiency, patterns have been refined by use.

o Look to the user and the purpose of an individual design above any best practices.

Navigation

• Good design makes it clear how users can move through and use application features.
– Annunciator Panel
– Fixed Menu
– Expandable Menu
– Scroll
– Notifications and Feedback

Android has a diverse ecosystem, with fewer institutionalized restrictions and a wider variety of mobile devices than other popular systems.

o They are strong competitor in the mobile market, but the flexibility of Android design can introduce new issues.

o Development of the Android operating system is led by Google

o 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://eyes-free.googlecode.com/svn/trunk/documentation/android_access/index.html).

iOS

o 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

o 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 non- standard 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.

BlackBerry OS

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

o 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

o **text-based** push-delivery messages,

o **closed captions** on multimedia content, and

o **hearing-aid compatibility** for hearing accessibility issues.

o **Low-vision users** can use the Clarity theme and other screen adjustments, and benefit from tactile keyboards.

o **Predictive text** and **AutoText** aid users with mobility and cognitive issues.

o Best practices and device capabilities are maintained online at http://docs.blackberry.com/en/

Windows Phone7

o Developed by **Microsoft**, Windows Phone 7 (WP7) is a currently smaller contender, focused on consumer markets.

o Using the “Metro” theme, features are divided into “Live Tiles” that link to applications.

o Six dedicated hardware buttons (back, start, search, camera, power, and volume), at least 4 GB of Flash memory, and Assisted GPS.

 Interface Tips

o Windows Phone 7 interfaces are minimalist,

 Using empty space to lend clarity to the application.

o WP7 uses movement over gradients for on-screen elements to immerse users in the application experience.

o 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.eaving focus**.**

o Do not use a “back” button to navigate back the page stack.

o Panorama controls slide horizontally through panes, and pivot controls list panes users can visit.

o 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

o 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.

o Learn more about the basics of WP7 accessibility at http://www.microsoft.com/

Mobile Web Browsers

o If a mobile application sends users to a website, that website should be optimized for mobile browsers.

o Similarly, mobile web applications should follow key mobile design methods.

o 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

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

Q3 a) Explain features of android

As Android is open source and freely available to manufacturers for customization, there are no fixed hardware and software configurations. However, Android itself supports the following features:

- Storage : Uses SQLite, a lightweight relational database, for data storage.
- Connectivity : Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), WiFi, LTE, and WiMAX.
- Messaging: Supports both SMS and MMS.
- Web browser: Based on the open-source WebKit, together with Chrome's V8 JavaScript engine
- **Media support:** Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC

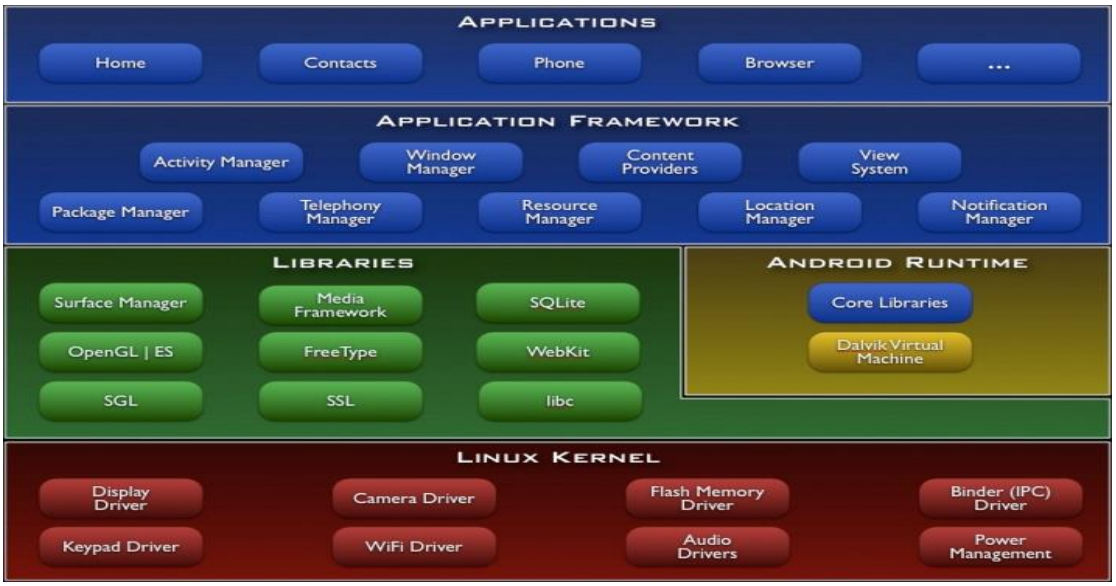
(in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP

- **Hardware support:** Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- **Multi-touch:** Supports multi-touch screens
- **Multi-tasking:** Supports multi-tasking applications
- **Flash support:** Android 2.3 supports Flash 10.1.
- **Tethering:** Supports sharing of Internet connections as a wired/wireless hotspot

Q3 b) Explain the various layers in android OS

- Android is a mobile operating system that is based on a modified version of Linux.
- It was originally developed by a start-up of the same name, Android, Inc.
- Android is open and free; most of the Android code was released under the open-source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code.
- The main advantage of adopting Android is that it offers a unified approach to application development.
- Android OS is a Linux-based open source platform for mobile, cellular handsets developed by Google and the Open Handset Alliance

The Android OS is roughly divided into five sections in four main layers as shown in Figure



Linux kernel: This is the kernel on which Android is based. This layer contains all the low level 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.

Q4 a) Briefly discuss the anatomy of 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:
  - **drawable** - **<resolution>**: All the image files to be used by the Android application must be stored here.
  - **layout -** contains **activity_main.xml** file, which the is GUI of the application.
  - **values -** contains files like **strings.xml, styles.xml** that are need 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_na
                me" >
                <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 :

- It defines the package name of the application as ***net.learn2develop.HelloWorld***.
- 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.
- The version name of the application is 1.0. This string value is mainly used for display to the user.
- The application uses the image named ***ic_launcher.png*** located in the ***drawable*** folder.
- The name of this application is the string named **app_name** defined in the ***strings.xml*** file.
- 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.
- 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.
- 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:

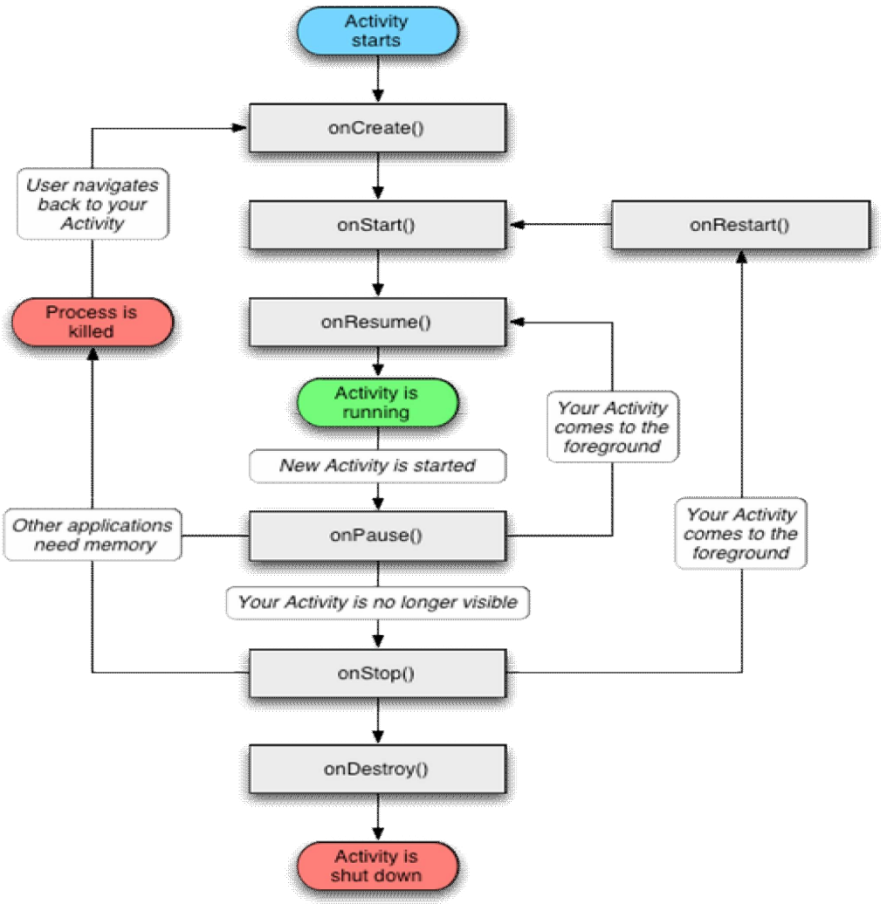
```
package net.learn2develop.HelloWorld;
import android.app.Activity;
import android.os.Bundle;

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.

Q4b) Discuss life cycle of an activity with an example code

The Activity base class defines a series of events that governs the life cycle of an activity. Figure 2.2 shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.



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 or by the system to conserve memory)
- onRestart() — Called when the activity has been stopped and is restarting again

By default, the activity created for you contains the onCreate() event. This event handler contains the code that helps to display the UI elements of your screen.

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity
{
        String tag = "Events";
```

```
/** Called when the activity is first created. */ @Override
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); Log.d(tag,
        "In the onCreate() event");
}

public void onStart()
{
        super.onStart();
        Log.d(tag, "In the onStart() event");
}
public void onRestart()
{
        super.onRestart();
        Log.d(tag, "In the onRestart() event");
}

public void onResume()
{
        super.onResume();
        Log.d(tag, "In the onResume() event");
}
public void onPause()
{
        super.onPause();
        Log.d(tag, "In the onPause() event");
}
public void onStop()
{
        super.onStop();
        Log.d(tag, "In the onStop() event");
}
public void onDestroy()
{
        super.onDestroy();
        Log.d(tag, "In the onDestroy() event");
}

}
```

Q5 a) Define a view Explain the different types of views

A view is a widget that has an appearance on screen. Basic views used to design the UI of your Android applications. These basic views enable you to display text information, as well as perform some basic selection.

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

TextView

The TextView view is used to display text to the user.

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

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

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

Button
Represents a push-button widget

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

ImageButton
Similar to the Button view, except that it also displays an image

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

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

```
<CheckBox android:id="@+id/chkAutosave"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Autosave" />
```

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.

```
<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 — Displays checked/unchecked states using a light indicator

```
<ToggleButton android:id="@+id/toggle1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Q5 b) Explain various view groups with suitable code segment

Android supports the following ViewGroups:
- LinearLayout
- AbsoluteLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

Each of these ViewGroups are explained hereunder.

LinerLayout

 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>
```

There is a problem with absolute layout when the activity is viewed on a high resolution screen. For this reason the AbslouteLayout has been deprecated since Android 1.5. It is not guaranteed to be supported in future version of android.

TableLayout

 The TableLayout groups views into rows and columns.

 <TableRow> element to designate a row in the table. Each row can contain one or more views.

 Each view placed within a row forms a cell.

 The width of each column is determined by the largest width of each cell in that column.

Consider the content of main.xml shown here:

```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_height="fill_parent" android:layout_width="fill_parent">
```

```
<TableRow>
<TextView
android:text=”User Name:” android:width =”120px” />
<EditText
android:id=”@+id/txtUserName” android:width=”200px” />
</TableRow>
<TableRow>
<TextView
android:text=”Password:” />
<EditText
android:id=”@+id/txtPassword” android:password=”true” />
</TableRow>
<TableRow>
<TextView />
<CheckBox android:id=”@+id/chkRememberPassword” android:layout_width=”fill_parent”
android:layout_height=”wrap_content” android:text=”Remember Password”/>
</TableRow>
<TableRow>
<Button
android:id=”@+id/buttonSignIn” android:text=”Log In” />
</TableRow>
</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other. Consider the following main.xml file:

```
<?xml version=”1.0” encoding=”utf-8”?>
<RelativeLayout android:id=”@+id/RLayout” android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=”http://schemas.android.com/apk/res/android”
>
<TextView android:id=”@+id/lblComments” android:layout_width=”wrap_content”
android:layout_height=”wrap_content” android:text=”Comments”
android:layout_alignParentTop=”true” android:layout_alignParentLeft=”true”
/>
<EditText android:id=”@+id/txtComments” android:layout_width=”fill_parent”
android:layout_height=”170px” android:textSize=”18sp”
android:layout_alignLeft=”@+id/lblComments” android:layout_below=”@+id/lblComments”
android:layout_centerHorizontal=”true”
/>
<Button android:id=”@+id/btnSave” android:layout_width=”125px”
android:layout_height=”wrap_content” android:text=”Save”
android:layout_below=”@+id/txtComments” android:layout_alignRight=”@+id/txtComments”
/>
<Button android:id=”@+id/btnCancel” android:layout_width=”124px”
android:layout_height=”wrap_content” android:text=”Cancel”
android:layout_below=”@+id/txtComments” android:layout_alignLeft=”@+id/txtComments”
/>
```

</RelativeLayout>
The UI of the above code would look like –
 Each view is embedded within the relative layout has attributes that enable it to align with another view.
 The value for each of these attributes is the **ID** for the view that you are **referencing.**
 These **attributes** are as follows:
o layout_alignParentTop
o layout_alignParentLeft
o layout_alignLeft
o layout_alignRight
o layout_below
o layout_centerHorizontal

FrameLayout

The FrameLayout is a placeholder on screen that you can use to **display** a single view. Views that you add to a FrameLayout are **always anchored** to the top left of the layout.

```
<RelativeLayout android:id=”@+id/RLayout” android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=”http://schemas.android.com/apk/res/android”
>
<TextView android:id=”@+id/lblComments” android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:text=”This is my lovely dog, Ookii” android:layout_alignParentTop=”true”
android:layout_alignParentLeft=”true”
/>
<FrameLayout android:layout_width=”wrap_content” android:layout_height=”wrap_content”
android:layout_alignLeft=”@+id/lblComments” android:layout_below=”@+id/lblComments”
android:layout_centerHorizontal=”true”
>
<ImageView
android:src = “@drawable/ookii” android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
/>
</FrameLayout>
</RelativeLayout>
```

Here, you have a FrameLayout within a RelativeLayout. Within the FrameLayout, you embed an ImageView. If you add another view (such as a Button view) within the FrameLayout, the view will overlap the previous view.

ScrollView

A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.

NOTE:

 Do not use a ListView together with the ScrollView.
 The ListView is designed for showing a list of related information and is optimized for dealing with large lists

```
<ScrollView
android:layout_width="fill_parent" android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android" >
<LinearLayout
android:layout_width="fill_parent" android:layout_height="wrap_content"
android:orientation="Vertical">
</LinearLayout>
</ScrollView>
```

Q6 a) How do you display Google maps in android application explain with code?

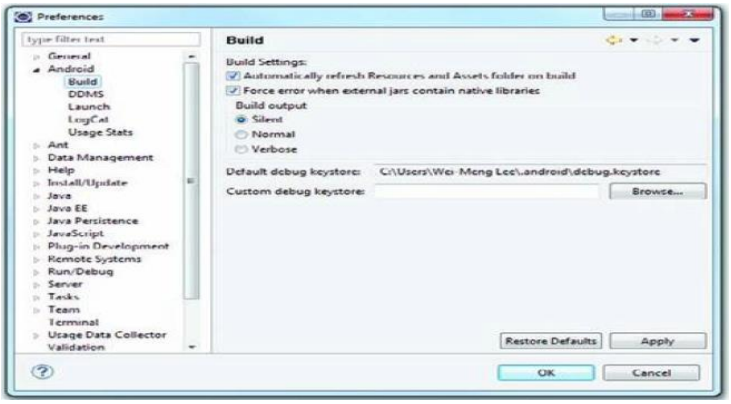
Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things. This section describes how to use Google Maps in your Android applications and programmatically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

Q6 b) Discuss APK file deployment in detail

Once you have signed your APK fi les, you need a way to get them onto your users' devices. Three methods are here:

- Deploying manually using the adb.exe tool
- Hosting the application on a web server
- Publishing through the Android Market

Besides the above methods, you can install your applications on users' devices through emails, SD card, etc. As long as you can transfer the APK file onto the user's device, you can install the application.

Using the adb.exe Tool:
Once your Android application is signed, you can deploy it to emulators and devices using the adb.exe (Android Debug Bridge) tool (located in the platform-tools folder of the Android SDK). Using the command prompt in Windows, navigate to the "<Android_SDK>\platform-tools" folder. To install the application to an emulator/device (assuming the emulator is currently up and running or a device is currently connected), issue the following command:

adb install "C:\Users\Wei-Meng Lee\Desktop\LBS.apk"
(Note that, here, LBS is name of the project)

Besides using the adb.exe tool to install applications, you can also use it to remove an installed application. To do so, you can use the shell option to remove an application from its installed folder:

adb shell rm /data/app/net.learn2develop.LBS.apk

Another way to deploy an application is to use the DDMS tool in Eclipse. With an emulator (or device) selected, use the File Explorer in DDMS to go to the /data/app folder and use the "Push a file onto the device" button to copy the APK file onto the device.

Using a Web Server:
If you wish to host your application on your own, you can use a web server to do that. This is ideal if you have your own web hosting services and want to provide the application free of charge to your users or you can restrict access to certain groups of people. Following are the steps involved:

- Copy the signed LBS.apk file to c:\inetpub\wwwroot\. In addition, create a new HTML file named Install.html with the following content:

```
<html>
    <title>Where Am I application</title>
    <body>
        Download the Where Am I application <a href="LBS.apk">here</a>
    </body>
</html>
```

- On your web server, you may need to register a new MIME type for the APK file. The MIME type for the .apk extension is application/vnd.android.packagearchive.
- From the Application settings menu, check the "Unknown sources" item. You will be prompted with a warning message. Click OK. Checking this item will allow the Emulator/device to install applications from other non-Market sources (such as from a web server).
- To install the LBS.apk application from the IIS web server running on your computer, launch the Browser application on the Android Emulator/device and navigate to the URL pointing to the APK file. To refer to the computer running the emulator, you should use the special IP address of 10.0.2.2.
- Alternatively, you can also use the IP address of the host computer. Clicking the "here" link will download the APK file onto your device. Drag the notification bar down to reveal the download status. To install the downloaded application, simply tap on it and it will show the permission(s) required by this application.
- Click the Install button to proceed with the installation. When the application is installed, you can launch it by clicking the Open button.

Besides using a web server, you can also

- email your application to users as a attachment; when the users receive the e-mail they can download the attachment and install the application directly onto their device.

Publishing on Android Market:

It is always better to host your application on Android market (Google Playstore). Steps involved in doing so, are explained hereunder:

- **Creating a Developer Profile:**
  - Create a developer profile at http://market.android.com/publish/Home using a Google account.
  - Pay one-time registration fees.
  - Agree Android Market Developer Distribution Agreement

- **Submitting Your Apps:**
  If you intend to charge for your application, click the Setup Merchant Account link located at the bottom of the screen. Here you enter additional information such as bank account and tax ID. You will be asked to supply some details for your application. Following are the compulsory details to be provided:
  - The application in APK format
  - At least two screenshots. You can use the DDMS perspective in Eclipse to capture screenshots of your application running on the Emulator or real device. A high-resolution application icon. This size of this image must be 512×512 pixels.
  - Provide the title of your application, its description and recent update details.
  - Indicate whether your application employs copy protection, and specify a content rating.

When all these setup is done, click Publish to publish your application on the Android Market.

Q7 a) Explain with code procedure to send SMS through android application

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.

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

Q7 b) Write and android application to download an image file from the web

The data may be binary data (an image file, for example), or text data etc. Consider the following example project which downloads an image file from the web.

 Use the same application that has been created in the previous section (as network connection has already been established).

 Add an ImageView within <LinearLayout> inside activity_main.xml file as follows:

```
<ImageView android:id=”@+id/img”
android:layout_width=”wrap_content” android:layout_height=”wrap_content”
android:layout_gravity=”center” />
```

 Add the following statements to the MainActivity.java file:

```
public class MainActivity extends Activity
{
ImageView img;
private InputStream OpenHttpConnection(String urlString)
throws IOException
{
//the code used in previous section
}
private Bitmap DownloadImage(String URL)
{
Bitmap bitmap = null; InputStream in = null; try
{
in = OpenHttpConnection(URL);
bitmap = BitmapFactory.decodeStream(in); in.close();
} catch (IOException e1)
{
Toast.makeText(this, e1.getLocalizedMessage(),
Toast.LENGTH_LONG).show(); e1.printStackTrace();
}
return bitmap;
}
/** Called when the activity is first created. */ @Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState); setContentView(R.layout.main);
//---download an image---
Bitmap bitmap = DownloadImage( “http://www.streetcar.org/mim/cable/images/cable-01.jpg”);
img = (ImageView) findViewById(R.id.img);
img.setImageBitmap(bitmap);
}
}
```

 Working procedure: The DownloadImage() method takes the URL of the image to download and then opens the connection to the server using the OpenHttpConnection() method that you have defined earlier. Using the InputStream object returned by the connection, the decodeStream() method from the BitmapFactory class is used to download and decode the data into a Bitmap object. The DownloadImage() method returns a Bitmap object.

 Run the application to see the following output:

Q8 a) Write an android application to download a text files from the web

One can download plain-text file from web. For example, you might be writing an RSS Reader application and hence need to download RSS XML feeds for processing. The following program shows how you can download a plain-text file in your application.

- Use the project created before (which has network connectivity).
- Write the following code in MainActivity.java file:

```
public class MainActivity extends Activity
{
ImageView img;
private InputStream OpenHttpConnection(String urlString)
throws IOException
{
//code used for network connectivity
}
private Bitmap DownloadImage(String URL)
{
//code used for downloading image/binary file
}
private String DownloadText(String URL)
{
int BUFFER_SIZE = 2000;
InputStream in = null; try
{
in = OpenHttpConnection(URL);
} catch (IOException e1)
{
Toast.makeText(this, e1.getLocalizedMessage(),
Toast.LENGTH_LONG).show(); e1.printStackTrace();
return "";
}
InputStreamReader isr = new InputStreamReader(in); int charRead;
String str = "";
char[] inputBuffer = new char[BUFFER_SIZE]; try
```

```
{
while ((charRead = isr.read(inputBuffer))>0)
{
//---convert the chars to a String--- String readString =
String.copyValueOf(inputBuffer, 0, charRead); str += readString;
inputBuffer = new char[BUFFER_SIZE];
}
in.close();
} catch (IOException e)
{
Toast.makeText(this, e.getLocalizedMessage(), Toast.LENGTH_LONG).show();
e.printStackTrace(); return “”;
}
return str;
}
/** Called when the activity is first created. */ @Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState); setContentView(R.layout.main);
//---download an image--- Bitmap bitmap =
DownloadImage( “http://www.streetcar.org/mim/cable/images/cable- 01.jpg”);
img = (ImageView) findViewById(R.id.img); img.setImageBitmap(bitmap);
//---download an RSS feed--- String str = DownloadText(
“http://www.appleinsider.com/appleinsider.rss”); Toast.makeText(getBaseContext(), str,
Toast.LENGTH_SHORT).show();
}
}
```

 Run the application to get the output as shown below –

Q8 b) Explain with code asynchronous calls in networking in view of android application

Connections discussed in the previous sections were all *synchronous* – that is, the connection to a server will not return until the data is received. In real life, this presents some problems due to network connections being inherently slow. When you connect to a server to download some data, the user interface of your application remains frozen until a response is obtained. In most cases, this is not acceptable. Hence, you need to ensure that the connection to the server is made in an asynchronous fashion.

The easiest way to connect to the server asynchronously is to use the AsyncTask class available in the Android SDK. Using AsyncTask enables you to perform background tasks in a separate thread and then return the result in a UI thread. Using this class enables you to perform background operations without needing to handle complex threading issues.
Using the previous example of downloading an image from the server and then displaying the image in an ImageView, you could wrap the code in an instance of the AsyncTask class, as shown below:

```
public class MainActivity extends Activity
{
ImageView img;
private class BackgroundTask extends AsyncTask
<String, Void, Bitmap>
{
protected Bitmap doInBackground(String... url)
{
Bitmap bitmap = DownloadImage(url[0]); return bitmap;
}
protected void onPostExecute(Bitmap bitmap) { ImageView img = (ImageView)
findViewById(R.id.img); img.setImageBitmap(bitmap);
}
}
private InputStream OpenHttpConnection(String urlString) throws IOException
{
...
}
```

Basically, you defined a class that extends the AsyncTask class. In this case, there are two methods within the BackgroundTask class — doInBackground() and onPostExecute(). You put all the code that needs to be run asynchronously in the doInBackground() method. When the task is completed, the result is passed back via the onPostExecute() method. The onPostExecute() method is executed on the UI thread, hence it is thread safe to update the ImageView with the bitmap downloaded from the server.
To perform the asynchronous tasks, simply create an instance of the BackgroundTask class and call its execute() method:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState); setContentView(R.layout.main);
new BackgroundTask().execute(
“http://www.streetcar.org/mim/cable/images/cable-01.jpg”);
}
```

Q9 a) Discuss components of iphone SDK

xCode and IOS SDK

To create native iOS applications, you will need to install both the xCode IDE as well as the iOS SDK. Although you can obtain xCode by using the App Store within Mac OS X. Downloading xCode and the SDK from the downloads section in the iOS Dev Center.
 Installation: After you follow the steps to install xCode, you should have the xCode IDE as well as a great deal of other useful development tools installed to

/Developer/Applications. You can start xCode from this directory or by using spotlight.
 Components of iPhone SDK: The iPhone SDK includes a great number of tools that help create iOS for apps. These tools range from debugging and profiling to developing. Following is the list of most common tools that we use that are included in the iOS SDK:
o **xCode:** xCode is Apple's Integrated Development Environment (IDE) for creating Objective-C applications. xCode enables you to manage, author, and debug your Objective-C projects.
o **Dashcode:** Dashcode is an IDE that enables you to develop web-based iPhone/iPad applications and Dashboard widgets.
o **iPhone Simulator:** This tool provides a method to simulate an iPhone or iPad device on your Mac, for use with testing your iOS applications.
o **Interface Builder:** The Interface Builder, or IB, is a visual editor that is used for designing the user interface for your iOS application.
o **Instruments:** Instruments is a suite of tools that helps you analyze your iOS application and monitor for performance bottlenecks as well as memory leaks in real time while attached to an iOS device or iOS Simulator.

Q9 b) Discuss anatomy of a iOS App

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.

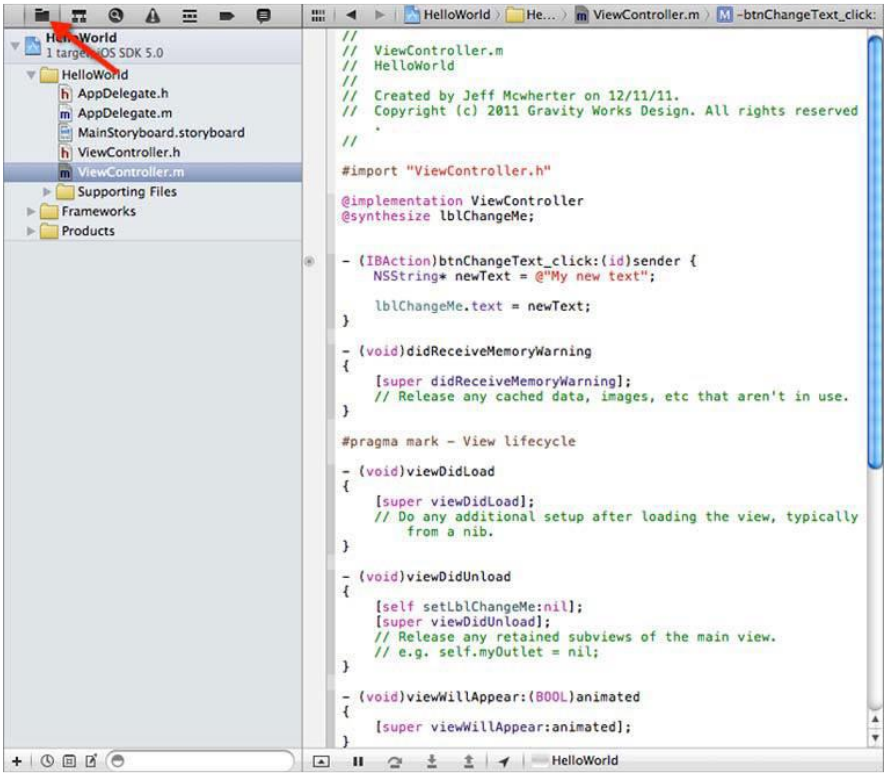
- **Views:** iPhone apps are made up of one or more views. Views usually have GUI elements such as text fields, labels, buttons, and so on. You can build a view built using the Interface Builder tool, which enables you to drag and drop controls on the view, or you can create a view entirely with code.
- **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.
- **Main.m:** As with any C program, the execution of Objective-C applications start from the main() function, which is the main.m file.
- **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.
- **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.

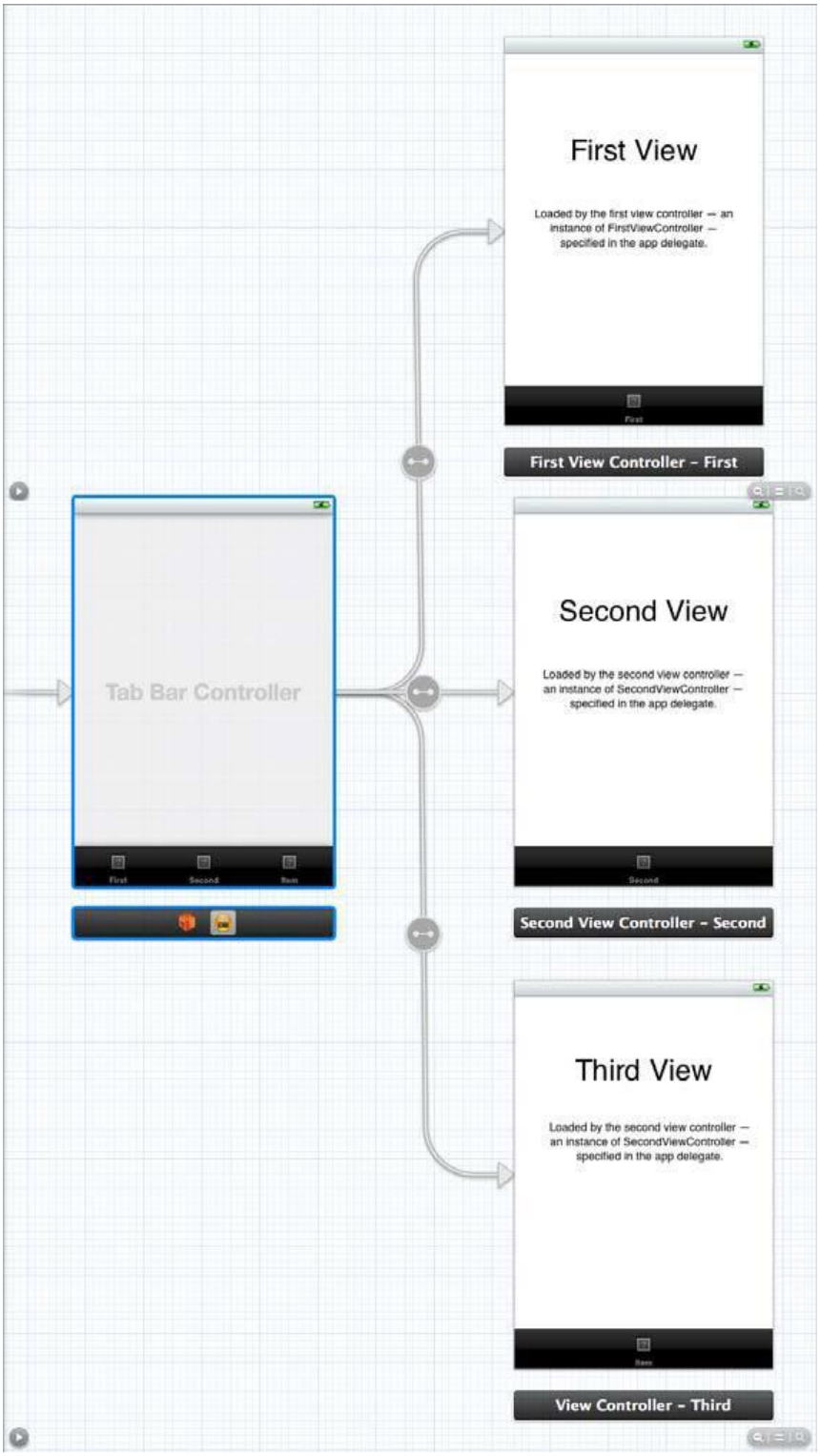
- **Supporting Files:** The supporting files directory contains files such as the plist setting files (which contain customizable application settings), as well as string resource files that are used within your app.

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.





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.

Q10 a) Discuss anatomy of a windows phone 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

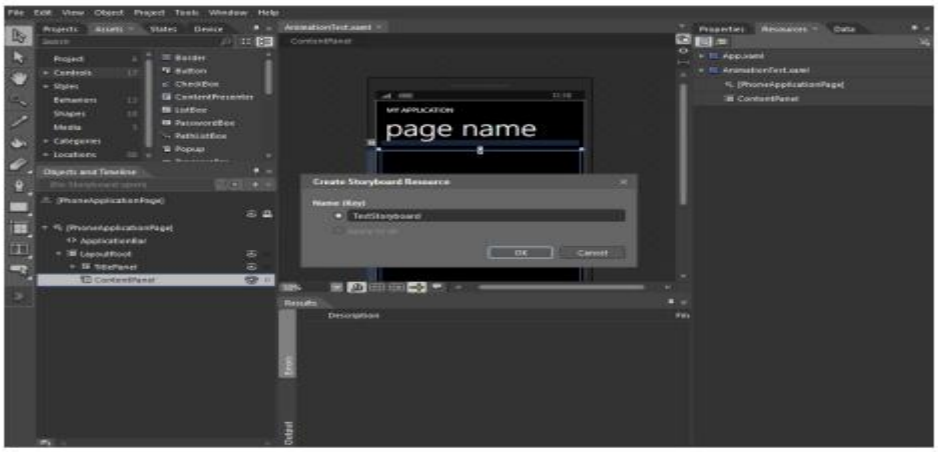


FIGURE 8-9: Storyboards in Blend

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.

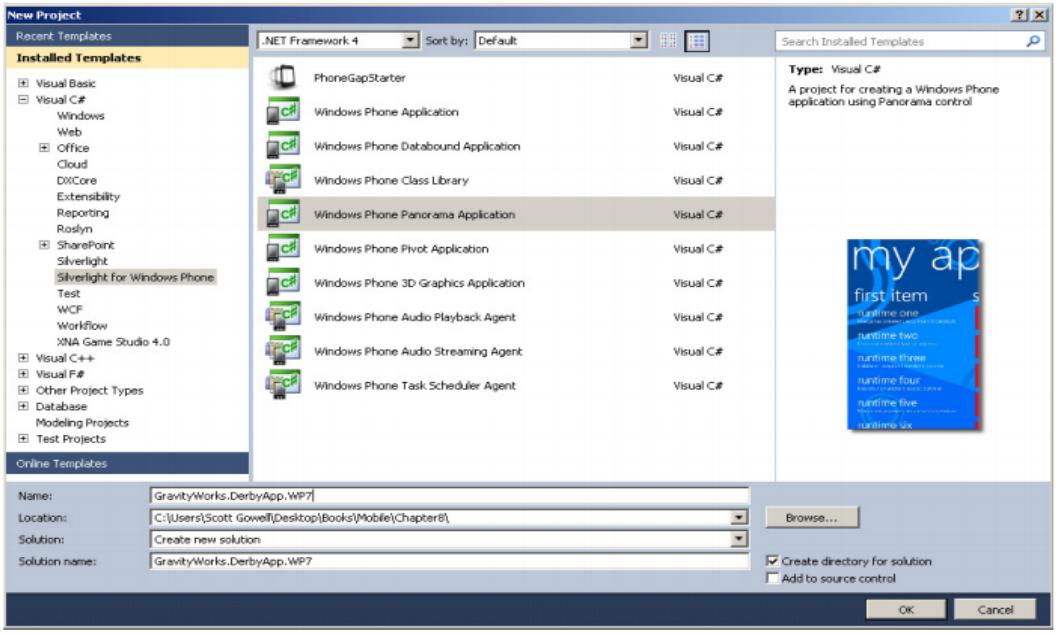
Q10b) Briefly explain various steps involved in developing derby App in windows phone 7

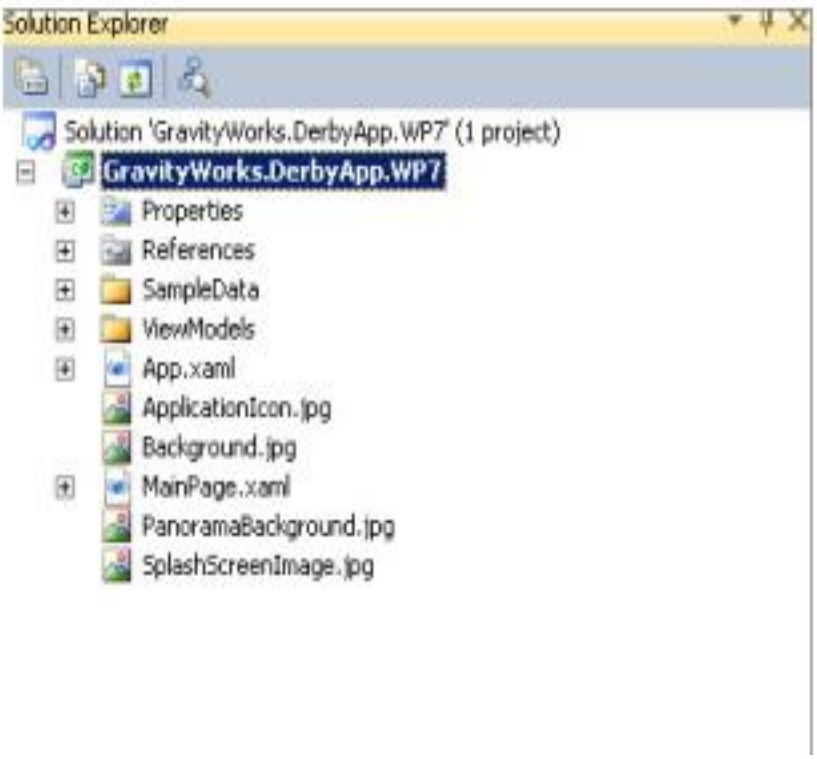
Here, you implement the features of the Derby Names project using Microsoft Visual Studio, while also taking time to learn Windows Phone 7–specific technologies.

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.

Q10 c) Write a short note on notification in windows phone 7 app and accelerometer in windows phone 7 app.

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.