CBCS SCHEME

20MCA263

USN 1CR21MC099

Second Semester MCA Degree Examination, July/August 2022
Mobile Applications Development

Time: 3 hrs. Max. Marks: 100

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

Module-1

1 a. What are the preliminary considerations for Mobile Application Development? **(10 Marks)**
b. Demonstrate effective use of Screen Rent Estate. **(06 Marks)**
c. List out any 4 features of Android OS, with respect to user interface design. **(04 Marks)**

OR

2 a. Discuss briefly on the design patterns for Navigation used in mobile user interface design. **(10 Marks)**
b. Explain Gestalt Principles **(06 Marks)**
c. Compare Simulators and Emulators. **(04 Marks)**

Module-2

3 a. Explain the basic components of an Android Application **(06 Marks)**
b. Write short notes on the anatomy of an Android Application. **(10 Marks)**
c. What is a fragment? How is it useful in designing an application? **(04 Marks)**

OR

4 a. What is an action bar? Demonstrate how to utilize it with an example. **(08 Marks)**
b. What is Android Manifest File? Explain its contents in detail. **(08 Marks)**
c. List out the tools required for developing an Androids application. **(04 Marks)**

Module-3

5 a. Illustrate the use of a list view with a code snippet. **(10 Marks)**
b. With an example application demonstrate the use of Checkbox, Radiobutton, Button EditText and TextVeiw. **(10 Marks)**

OR

6 a. Explain the steps involved in displaying maps in an android application. **(10 Marks)**
b. Discuss on the steps involved in publishing an android application in Google Playstore. **(10 Marks)**

Module-4

7 a. Define service? Explain how an activity can invoke a service with an example. **(10 Marks)**
b. Give the code snippet for downloading an image from a given URL and display it using ImageView. **(10 Marks)**

OR

8 a. Explain briefly how SMS works in Android. **(10 Marks)**
b. Write short notes on Asynchronous tasks in Android. **(08 Marks)**
c. Name the permissions you need to declare in your Android Manifest file for HTTP connection. **(02 Marks)**

Module-5

9	a.	Explain briefly the steps to create native IoS applications.	(10 M
	b.	Discuss on the anatomy of an IoS application.	(10 M

OR

10	a.	Explain the various types of Notifications used in Windows Phone 7.	(08 M
	b.	Write short note on the following with respect to windows Phone 7. i) Offline Storage ii) Accelerometer.	(08 M
	c.	Explain GPS features for Windows Phone 7 Application.	(04 M

* * * * *

1.a. What are the preliminary considerations for Mobile Application Development?

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. Demonstrate effective use of screen real estate.

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

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

1.c. List out any 4 features of Android Os with respect to user interface design.

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

2.a. Discuss briefly on the design patterns for navigation used in mobile user interface design .

Design Patterns

A design pattern recycles and repurposes components, reusing the best ideas. More than time efficiency, patterns have been refined by use. Look to patterns, and maintain a pattern library that works for you, but 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.

1. Annunciator Panel

An annunciator panel, seen at the top of Figure, gives information on the state of a mobile device. Though each mobile device will provide a slightly different panel, developers can modify or suppress the annunciator panel — which lists the hardware features such as network connection and battery power — within an application. Because the information in this area is only notifications, application users will not usually have any direct interaction with the annunciator panel.

2. Fixed Menu

A menu that remains fixed to the viewport as users roam content is useful in many situations:

➢ When users need immediate access to frequently selected functionality on multiple screens

➢ When a revealable menu is already used on the same screen

➢ When a lack of controls, conflict with key interactions, or low discovery makes a revealable menu a poor choice

Because fixed menus are always accessible, users can interact with page content as needed; keep in mind the small screen real estate, and limit any fixed menus to the absolute necessities. Do not stack multiple fixed menus on top of each other, and reconsider the size and scope of an application if you have a fixed menu at both the top and bottom of a screen. The style, such as whether the menu goes across the top or bottom of the viewport, will be largely determined by the mobile device.

3. Expandable Menu

When all function options for one page cannot fit on the viewport, an expanding menu can

provide selectively available options: a full menu similar to the one shown in Figure will reveal once prompted. A gesture, like a swipe or double tap, may prompt the reveal as well as selecting an on-screen icon. Soft keys — hardware buttons connected to on-screen labels — may also prompt a menu to reveal.

Users often benefit from multiple access options, especially for the harder-to-find gestural prompts. Users may exit a menu using a back button, a close button that is part of the revealed menu list, or by toggling with the same gesture or soft key that prompted the reveal. Try to keep close functionality on the screen while the menu is revealed.

Scroll

• It is best to limit scrolling, limiting application screens to the size of the viewport whenever possible. Some types of apps you develop will require scrolling to present data to the user effectively, such as an email client. When scrolling must occur, check that the design communicates the area that can be scrolled, the current context of a user, and the ratio of the current view to the total content available

• Only in-focus items should be able to scroll. Make an application more immersive, incorporating gestures such as tilting to scroll through content. Make sure that scrolling takes place only on a single axis, if possible. When scrolling must occur both horizontally and vertically, consider providing a thumbnail of the user's place within the entire scrolling area.

• The vertical list simply displays textual information, and is the foundation of information display on many mobile devices.

• Graphical data — profile photos, category icons, status indicators — can clarify content lists. Use position as well as size, contrast, and other identifiers to show the visual importance of elements that users utilize to sort information.

• Thumbnails can be replaced by a default icon when a specific image is not available, but the value of the graphical indicator diminishes as more icons are not individually identifiable. Use icons and images to emphasize clarity and categorical distinction — embracing a strong and varied silhouette — over personality or generic graphics.

• An expandable list, reveals additional, valuable content for selected (touched) list items without leaving the current view.

• New information can be revealed as an animation whenever possible, aligning users

with the structure and context of the expanded content areas. Higher-priority information, whether the revealed content or the list item title, should be set apart with size, color, or contrast.

• Users may scroll by gesture, device tilt, or on-screen buttons. Choose a different pattern if live-scrolling — a pixel-by-pixel response to user input — is not possible.

5. Notifications and Feedback

If the user needs new information, application creators must use notifications. These prompts pause ongoing tasks, and enable users to change processes and behaviors according to the new information or feedback. Notification can inform a user (presenting a single button to close the notification), offer the ability to cancel a process or correct an error, or give multiple options from which to select. A user should never be presented with more than three options from any notification

2.b. Explain Gestalts principles.

Gestalt principles include proximity, closure, continuity, figure and ground, and similarity. **Proximity**

FIGURE 4-1: Proximity

• Users tend to group objects together. Elements placed near each other are perceived in groups; as shown in Figure, people will see one group of three gears, and one group of two gears. • 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

FIGURE 4-2: 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. • Figure illustrates the concept of closure: people recognize a triangle even though the figure is not complete.

- Harness the closure concept to create icons with a strong primary silhouette, without overloading users on pixilated and overdone details.
- In grid patterns with horizontal and vertical visual lines, use closure to precisely show the inside and outside of list items.

Continuity

FIGURE 4-3: 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.
- When people see Figure 4-3, they perceive the horizontal stroke as distinct from the curled stroke, even though these separate elements overlap.

- 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

FIGURE 4-4: Figure and ground

- A figure, such as a letter on a page, is surrounded by white space, or the ground. In Figure 4-4, the figure is the gear icon, and the ground is the surrounding space.
- Complex designs can play with the line between "figure" and "ground," but mobile interfaces speed user frustration with unclear distinctions.
- Primary controls and main application content should maintain a distinct separation between figure and ground.

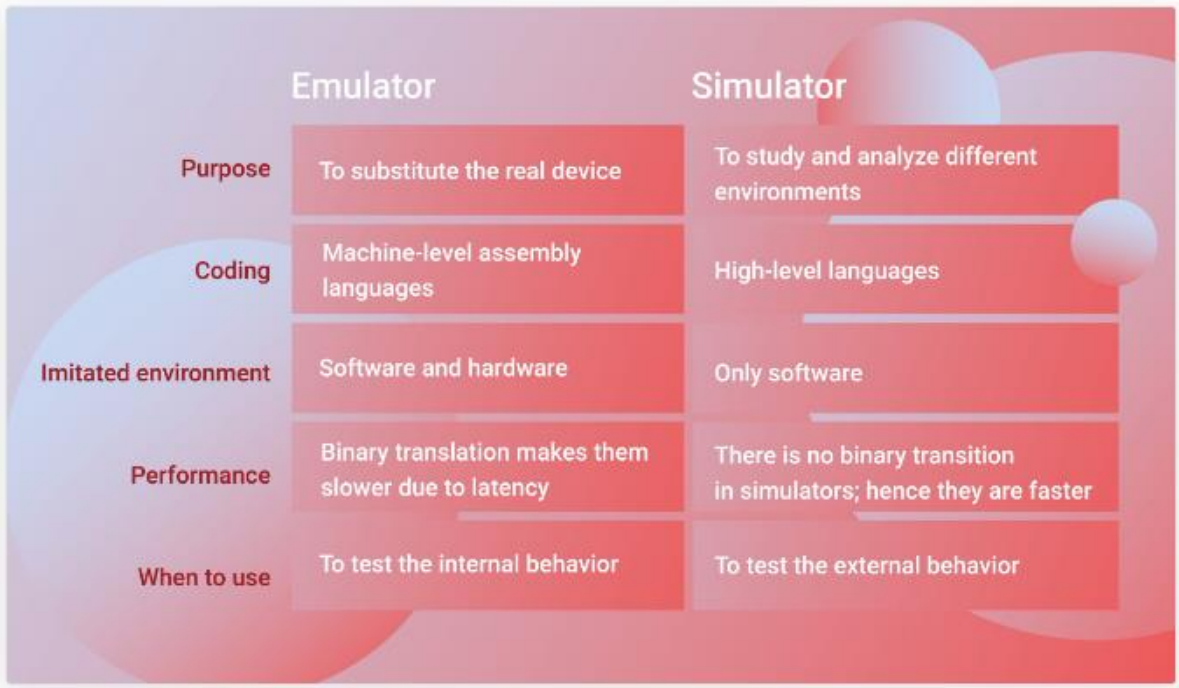
Similarity

FIGURE 4-5: Similarity

- Similar elements are grouped in a semi automated manner, according to the strong visual perception of color, form, size, and other attributes (see Figure).
  - 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.

2.c. Compare simulators and emulators.



3.a. Explain the basic 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

3.b. Write a short notes on 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:

o 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.

o 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:
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.javaYou 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.

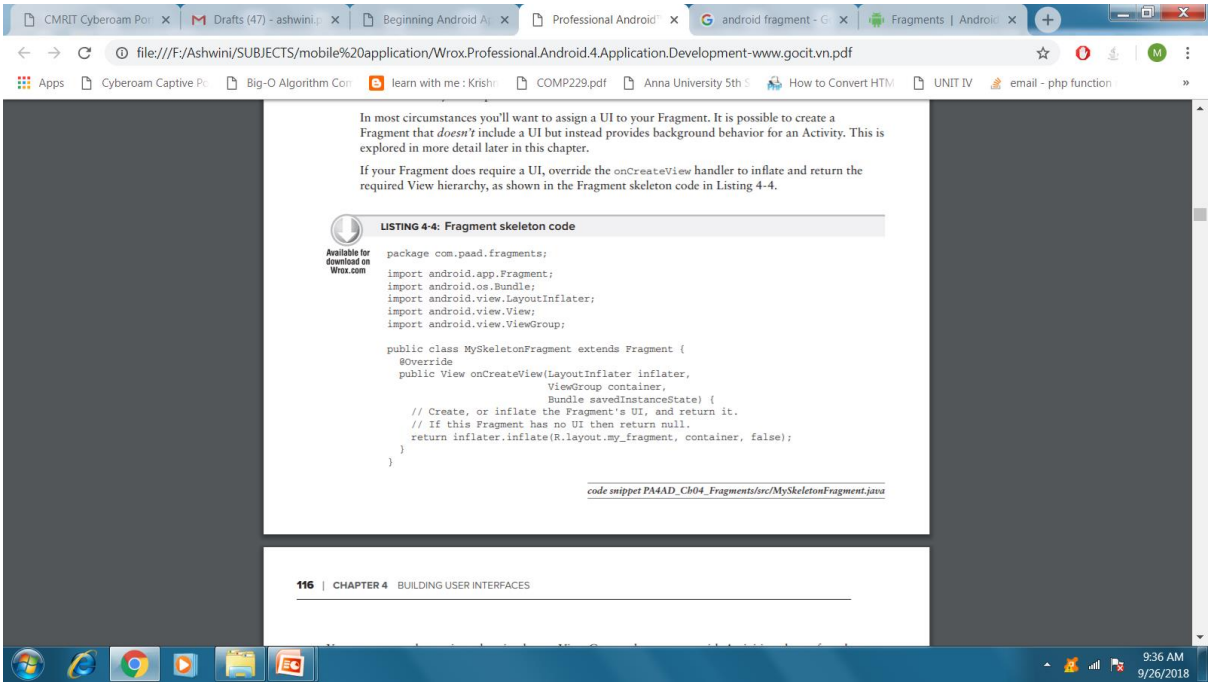
3.c. What is a fragment? How is it useful in designing an application?
Fragments enable you to divide your Activities into fully encapsulated reusable components, each with its own lifecycle and UI.

The primary advantage of Fragments is the ease with which you can create dynamic and flexible UI designs that can be adapted to suite a range of screen sizes — from small-screen smartphones to tablets.

Each Fragment is an independent module that is tightly bound to the Activity into which it is placed. Fragments can be reused within multiple activities, as well as laid out in a variety of combinations to suit multipane tablet UIs and added to, removed from, and exchanged within a running Activity to help build dynamic UIs.

Fragments provide a way to present a consistent UI optimized for a wide variety of Android device types, screen sizes, and device densities.

Although it is not necessary to divide your Activities (and their corresponding layouts) into Fragments, doing so will drastically improve the flexibility of your UI and make it easier for you to adapt your user experience for new device configurations.
Extend the Fragment class to create a new Fragment



In most circumstances you'll want to assign a UI to your Fragment. It is possible to create a Fragment that *doesn't* include a UI but instead provides background behavior for an Activity. This is explored in more detail later in this chapter.

If your Fragment does require a UI, override the `onCreateView` handler to inflate and return the required View hierarchy, as shown in the Fragment skeleton code in Listing 4-4.

LISTING 4-4: Fragment skeleton code

Available for download on Wrox.com

```
package com.paad.fragments;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class MySkeletonFragment extends Fragment {
  @Override
  public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
    // Create, or inflate the Fragment's UI, and return it.
    // If this Fragment has no UI then return null.
    return inflater.inflate(R.layout.my_fragment, container, false);
  }
}
```

code snippet PA4AD_Ch04_Fragments/src/MySkeletonFragment.java

You can create a layout in code using layout View Groups; however, as with Activities, the preferred way to design Fragment UI layouts is by inflating an XML resource.

Unlike Activities, Fragments don't need to be registered in your manifest. This is because Fragments can exist only when embedded into an Activity, with their lifecycles dependent on that of the Activity to which they've been added.

4.a. What is an action bar? Demonstrate how to utilize it with an example.

INTRODUCING THE ACTION BAR

The Action Bar component, shown in Figure, was introduced in Android 3.0 (API level 11). It's a navigation panel that replaces the title bar at the top of every Activity and that formalizes a common Android design pattern.

It's possible to hide the Action Bar, but best practice is to keep it and customize it to suit the style and navigation requirements of your application.

The Action Bar can be added to each Activity within your application and is designed to provide a consistent UI between applications and within a particular application's Activities.

The Action Bar provides a consistent framework for providing branding, navigation, and surfacing the key actions to be performed within an Activity. Although the Action

Bar provides a framework for presenting this functionality consistently across applications, the following sections describe how you can select which options are suitable for your application — and how they can be implemented.

Enabling and Disabling Action Bar

The Action Bar is enabled by default in any Activity that uses the (default) Theme.Holo theme and whose application has a target (or minimum) SDK version of 11 or higher.

To toggle the visibility of the Action Bar at run time, you can use its show and hide methods (In respective activity.java file):

```
import android.app.ActionBar;
ActionBar actionBar = getActionBar();
// Hide the Action Bar
actionBar.hide();
// Show the Action Bar
actionBar.show ();
```

Alternatively, you can apply a theme that doesn't include the Action Bar.

When you apply a theme that excludes the Action Bar from an Activity, you can't programmatically display it at run time. A call to getActionBar will return null.

You can create or customize your own theme that removes the Action Bar by setting the android:windowActionBar style property to false:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="NoActionBar" parent="@style/ActivityTheme">
<item name="android:windowActionBar">false</item>
</style>
</resources>
```

Customizing the Action Bar

Modifying the Icon and Title Text

1) Hide the title text (label)

```
ActionBar actionBar = getActionBar();
actionBar.setDisplayShowTitleEnabled(false);
```

2) Hide Logo and Icon

```
actionBar.setDisplayShowHomeEnabled(false);
```

3) Add Title and subtitle

```
actionBar.setSubtitle("Subtitle");
actionBar.setTitle("Title");
```

4) Change logo

```
// Makes logo Transparent
getActionBar().setIcon(new ColorDrawable(getResources().getColor(android.R.color.transparent)));
// Change logo to new image
getActionBar().setIcon(getResources().getDrawable(R.drawable.pizza));
```

Customizing the background

1) Change background color

```
import android.graphics.drawable.ColorDrawable;
import android.graphics.Color;
actionBar.setBackgroundDrawable(new ColorDrawable(Color.parseColor("#ff0000")));
```

2) Change background image

```
ActionBar actionBar = getActionBar();
Resources r = getResources();
Drawable myDrawable = r.getDrawable(R.drawable.gradient_header);
actionBar.setBackgroundDrawable(myDrawable);
```

OR

actionBar.setBackgroundDrawable(getResources().getDrawable(R.drawable.pizza));

Configuring Action Bar Icon Navigation Behavior

Android ActionBar Menu

1) The simplest way to get toolbar icons and action overflow items into the action bar is by creating menu XML resource file found in res/menu folder. We can add menu items in the raw xml file present in the folder as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
<!-- Search / will display always -->
<item android:id="@+id/action_search"
android:icon="@drawable/ic_launcher"
android:title="@string/action_search"
android:showAsAction="ifRoom"/>
<!-- Location Found -->
<item android:id="@+id/action_location_found"
android:icon="@drawable/ic_launcher"
android:title="@string/action_location_found"
android:showAsAction="ifRoom" />
<!-- Refresh -->
<item android:id="@+id/action_refresh"
android:icon="@drawable/ic_launcher"
android:title="@string/action_refresh"
android:showAsAction="ifRoom" />
<!-- Help -->
<item android:id="@+id/action_help"
android:icon="@drawable/ic_launcher"
```

```
android:title="@string/action_help"
android:showAsAction="never"/>
<!-- Check updates -->
<item android:id="@+id/action_check_updates"
android:icon="@drawable/ic_launcher"
android:title="@string/action_check_updates"
android:showAsAction="never" />
</menu>
```

2) There are four things that are needed to be configured for every menu item.

1. android:id: attribute specifies the id of the menu item. This works like ids anywhere else in the Android app. An android:id value starting with a @+id/ will create a constant in the R.menu constant collection
2. android:title: attribute value contains the title of the menu item
3. android:icon: attribute references an icon in the drawable directories
4. android:showAsAction: This attribute indicates how the given item should be portrayed in the action bar.

We can choose from any of the flags mentioned below:

- always to keep it in the ActionBar at all times
- ifRoom to keep it only if space is available
- never this means that the menu item will not be placed in the ActionBar as an icon. It will only be visible when the menu button is clicked, in the menu that's popping up
- withText : we can append this to either always or ifRoom, to indicate that the toolbar button to be both the icon and the title, not just the icon

Note that always is not guaranteed to be a toolbar button – if you ask for 100 always items, you will not have room for all of them. However, always items get priority for space in the action bar over ifRoom items.

3) Inflating the Menu Into the Android ActionBar

In order for the menu items defined in the menu XML file to be displayed, you need to

inflate the menu file. We do so inside the onCreateOptionsMenu() method of the activity where we wish to add the ActionBar. Here is the code snippet:

```
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar if it is present.
//getMenuInflater().inflate(R.menu.main, menu);
getMenuInflater().inflate(R.menu.action_menu, menu);
return true;
}
```

The R.menu.action_menu parameter is the constant referring to the menu XML file. The menu parameter is the menu into which we want to inflate the menu items.

4) Responding to Android Action Bar Events

To find out when the user taps on one of these things, we'll need to override onOptionsItemSelected()from the MainActivity as shown below:

```
public boolean onOptionsItemSelected(MenuItem item) {
TextView count;

// Take appropriate action for each action item click

switch (item.getItemId()) {
case R.id.action_search:
count=(TextView)findViewById(R.id.textview);
count.setText("Search is clicked");
return true;
case R.id.action_location_found:
// location found
count=(TextView)findViewById(R.id.textview);
count.setText("location found is clicked");
return true;
case R.id.action_refresh:
```

```
// refresh
count=(TextView)findViewById(R.id.textview);
count.setText("refresh is clicked");

return true;
case R.id.action_help:
// help action
count=(TextView)findViewById(R.id.textview);
count.setText("help is clicked");
return true;
case R.id.action_check_updates:
// check for updates action
count=(TextView)findViewById(R.id.textview);
count.setText("check updates is clicked");
return true;
default:
return super.onOptionsItemSelected(item);
}
}
```

4.b. What is android manifest file? Explain its contents in detail.

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.

4.c. List out the tools required for developing an Android application.

Eclipse (IDE)

For Android development, you should download the

Eclipse IDE for Java EE Developers (www.eclipse .org/downloads/packages/eclipse-ide-java-eedevelopers

/ heliossr1).

Once the Eclipse IDE is downloaded, unzip its content

(the eclipse folder) into a folder, say C:\Android\.

Android SDK

The Android SDK contains a debugger, libraries, an emulator, documentation, sample code, and tutorials.

You can download the Android SDK from http://developer.android.com/sdk/index.html.

Once the SDK is downloaded, unzip its content (the android-sdk-windows folder) into the C:\Android\ folder, or whatever name you have given to the folder you just created.

3) Android Development Tools (ADT)

The Android Development Tools (ADT) plug-in for Eclipse is an extension to the Eclipse IDE that supports the creation and debugging of Android applications.

Using the ADT, you will be able to do the following in Eclipse:

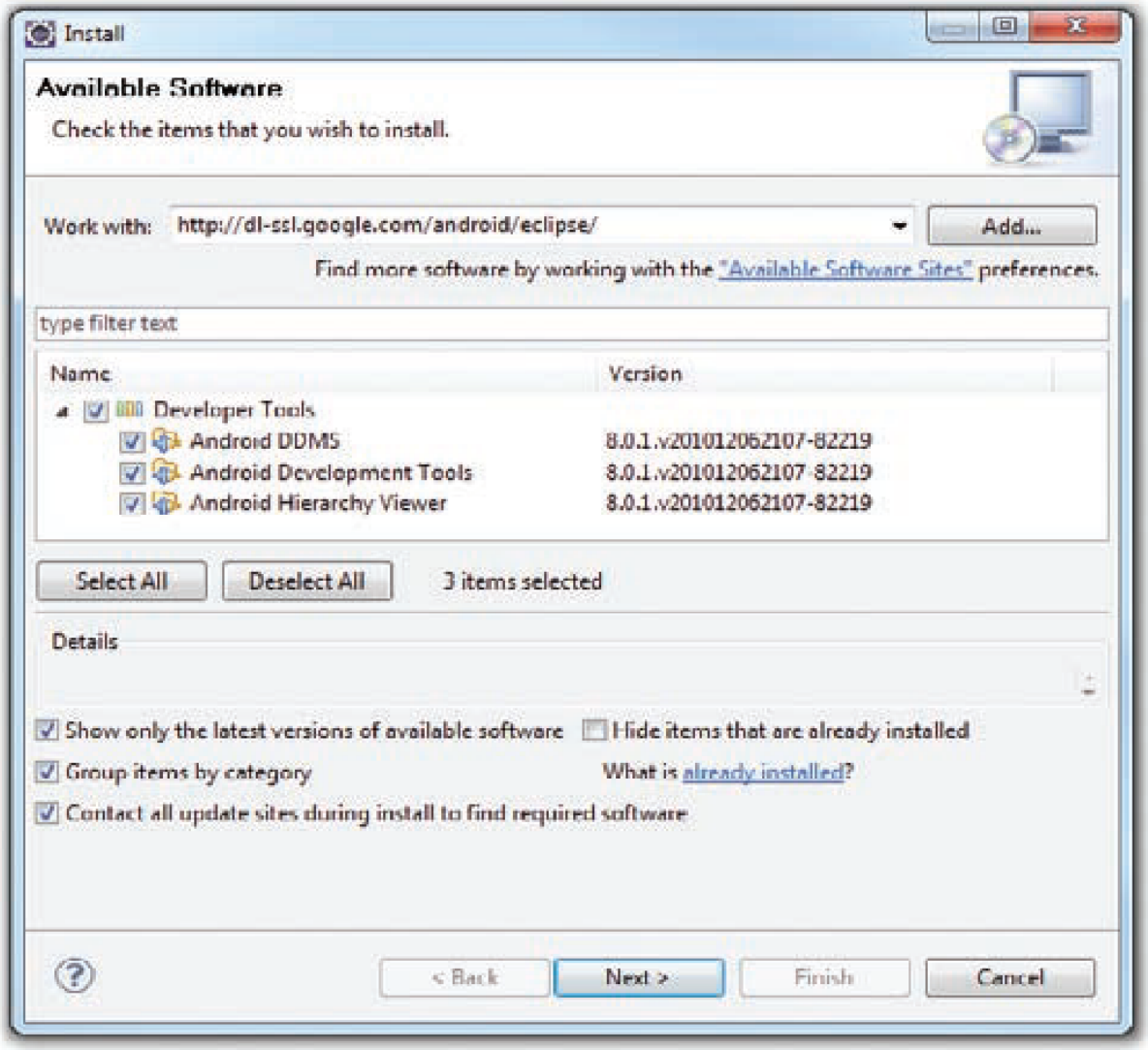
- ➤Create new Android application projects.
- ➤Access the tools for accessing your Android emulators and devices.
- ➤Compile and debug Android applications.
- ➤Export Android applications into Android Packages (APK).
- ➤Create digital certificates for code-signing your APK.

To install the ADT, first launch Eclipse by double-clicking on the eclipse.exe file located in the eclipse folder.

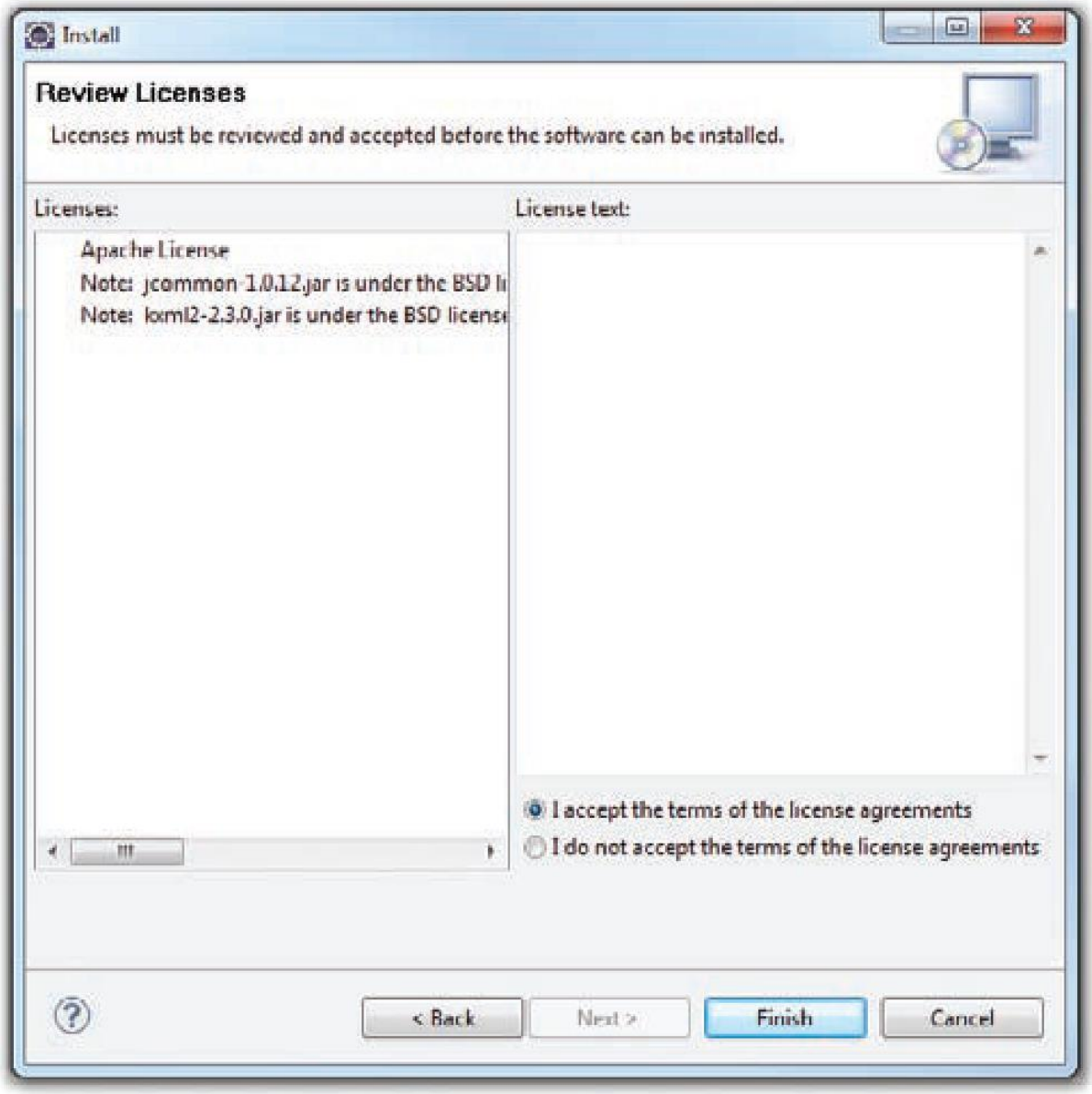
When Eclipse is first started, you will be prompted for a folder to use as your workspace.

In Eclipse, a workspace is a folder where you store all your projects. Take the default suggested and click OK.

Once Eclipse is up and running, select the Help ⇨Install New Software… menu item



You will be asked to review the licenses for the tools. Check the option to accept the license agreements.Click Finish to continue.



5.a. Illustrate the use of a list view with a code snippet.

Android **ListView** is a view which contains the group of items and displays in a scrollable list. ListView is implemented by importing *android.widget.ListView* class. ListView is a default scrollable which does not use other scroll view.

ListView uses Adapter classes which add the content from data source (such as string array, array, database etc) to ListView. Adapter bridges data between an *AdapterViews* and other Views (ListView, ScrollView etc).

File: activity_main.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="listview.example.com.listview.MainActivity">

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        />
</android.support.constraint.ConstraintLayout>
```

strings.xml

File:strings.xml

```
<resources>
    <string name="app_name">ListView</string>
    <string-array name="array_technology">
        <item>Android</item>
        <item>Java</item>
        <item>Php</item>
        <item>Hadoop</item>
        <item>Sap</item>
        <item>Python</item>
        <item>Ajax</item>
        <item>C++</item>
        <item>Ruby</item>
        <item>Rails</item>
        <item>.Net</item>
        <item>Perl</item>
    </string-array>
</resources>
```

File: MainActivity.java

```
package listview.example.com.listview;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
```

```
10. import android.widget.Toast;
11.
12. public class MainActivity extends AppCompatActivity {
13.     ListView listView;
14.     TextView textView;
15.     String[] listItem;
16.     @Override
17.     protected void onCreate(Bundle savedInstanceState) {
18.         super.onCreate(savedInstanceState);
19.         setContentView(R.layout.activity_main);
20.
21.         listView=(ListView)findViewById(R.id.listView);
22.         textView=(TextView)findViewById(R.id.textView);
23.         listItem = getResources().getStringArray(R.array.array_technology);
24.         final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
25.                 android.R.layout.simple_list_item_1, android.R.id.text1, listItem);
26.         listView.setAdapter(adapter);
27.
28.         listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
29.             @Override
30.             public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {
31.                 // TODO Auto-generated method stub
32.                 String value=adapter.getItem(position);
33.                 Toast.makeText(getApplicationContext(),value,Toast.LENGTH_SHORT).show();
34.
35.             }
36.         });
37.     }
38. }
```

5.b. With an example illustrate the use of Checkbox, Radio button, Button, Edit text and Text View.

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

   xmlns:tools="http://schemas.android.com/tools"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   tools:context="com.example.studytonightandroid.MainActivity" >


   <TextView
```

```
    android:id="@+id/tvRg"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:gravity="center"
    android:text="Rate This Lesson"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/tvRg"
    android:layout_centerHorizontal="true"
    android:orientation="horizontal"
    android:showDividers="beginning|middle|end"
    android:layout_marginTop="10dp"
    android:id="@+id/radioGroup" >


    <RadioButton
        android:id="@+id/rb1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="EXCELLENT"
        android:checked="false" />

    <RadioButton
        android:id="@+id/rb2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="GOOD"
        android:checked="true" />

    <RadioButton
        android:id="@+id/rb3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OKAY"
        android:checked="false" />

    <RadioButton
        android:id="@+id/rb4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="POOR"
        android:checked="false" />

  </RadioGroup>

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="SUBMIT"
    android:id="@+id/btnSubmit"
    android:layout_below="@+id/radioGroup"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"/>

</RelativeLayout>
```

Java File

```
package com.example.akshay.studytonightandroid;
```

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

   // These are the global variables
   RadioGroup radioGroup;
   RadioButton selectedRadioButton;
   Button buttonSubmit;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      // layout instances
      buttonSubmit = (Button) findViewById(R.id.buttonSubmit);
      radioGroup = (RadioGroup) findViewById(R.id.radioGroup);

      /*
         Submit Button
      */
      buttonSubmit.setOnClickListener(new View.OnClickListener() {

         @Override
```

```
        public void onClick(View v) {

            // get the selected RadioButton of the group

            selectedRadioButton                                                    =
(RadioButton)findViewById(radioGroup.getCheckedRadioButtonId());

            //get RadioButton text

            String yourVote = selectedRadioButton.getText().toString();

            // display it as Toast to the user

            Toast.makeText(MainActivity.this, "Selected Radio Button is:" + yourVote ,
Toast.LENGTH_LONG).show();

        }

      });

   }

}
```

6.a. Explain the steps involved in displaying maps in an android application.

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.•

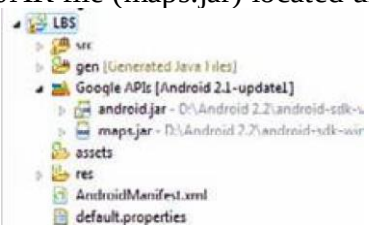
Obtain the latitude and longitude of locations in Google Maps.•

Perform geocoding and reverse geocoding (translating an address to latitude and• longitude and vice versa).

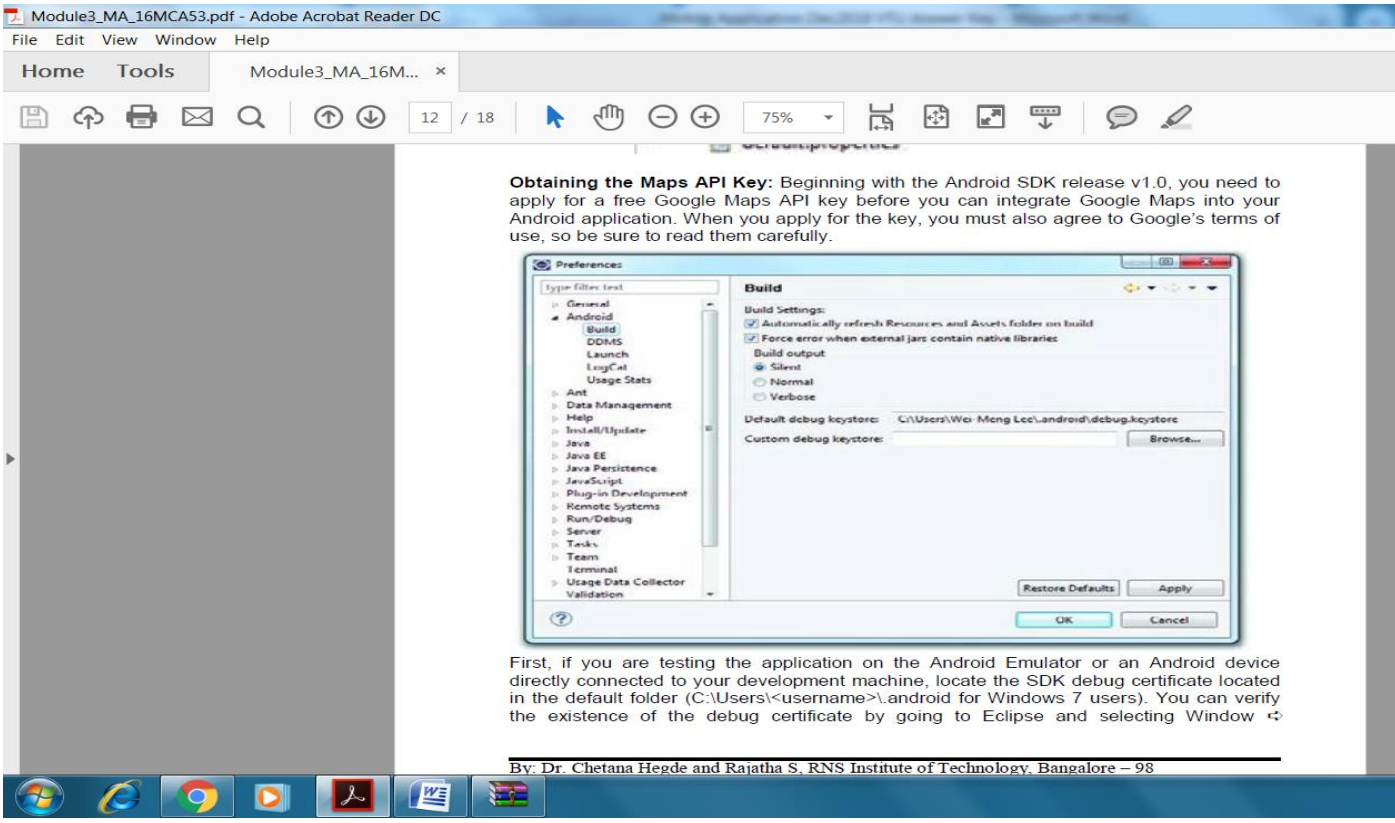
Add markers to Google Maps.•

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.

6.b. Discuss the steps involved in publishing an android application in Google Playstore.

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 fi le to c:\inetpub\wwwroot\.

```
<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 e-mail your application to users as an 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:

o Create a developer profile at http://market.android.com/publish/Home using a Google account.
o Pay one-time registration fees.

o 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:
o The application in APK format
o 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.
o Provide the title of your application, its description and recent update details.
o 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.

7.a. Define Service? Explain how an activity can invoke a service with an example.

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

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

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

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

- In the AndroidManifest.xml file, add the following statement :

```
<service android:name=".MyService" />
```

- In the activity_main.xml file, add the following statements in bold:

```
<Button android:id="@+id/btnStartService"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Start Service" />
<Button android:id="@+id/btnStopService"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Stop Service" />
```

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

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

7.b. Give the code snippet for downloading an image from a given URL and display it using Image View.

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:

8.a Explain briefly how SMS 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.

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);
}
}
```

Observe the sendSMS() method, which contains sendTextMessage() method. 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

8.b. Write short notes on Synchronous tasks in Android.

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



{
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”);

}
```

8.c. Name the permissions you need to declare in your Andriod Manifest file for HTTP connection

Add the following line in AndroidManifest.xml file:

<uses-permission

android:name="android.permission.INTERNET"><uses-permission/>;

9.a. Explain briefly the steps to create native IOS applications.

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

- Views: iPhone apps are made up of one or more views. Views usually have GUI elements such as text fields, labels, buttons, and so on. You can build a view built using the Interface Builder tool, which enables you to drag and drop controls on the view, or you can create a view entirely with code.
- Code that makes the Views work: Because iOS applications follow the MVC design pattern, there is a clean break between the UI and code that provides the application code.
- Resources: Every iOS application contains an icon file, an info.plist file that holds information about the application itself and the binary executable. Other resources such as images, sounds, and video are also classified as resources.
- Project Structure in Depth: When an iOS project is created within xCode, the IDE creates a set of files that are ready to run. These files provide the basics of what is needed to get going with a new project.

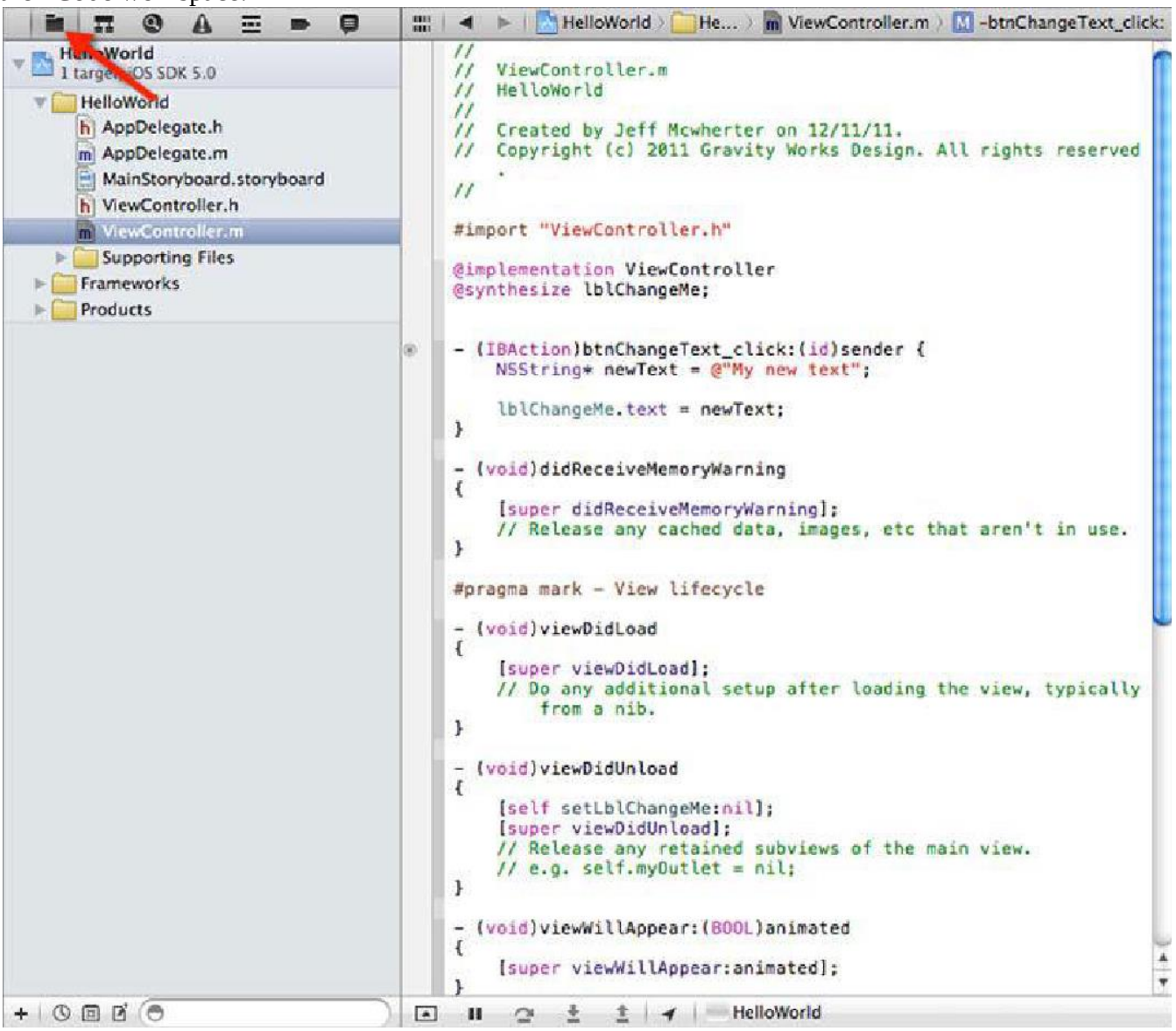
o Main.m: As with any C program, the execution of Objective-C applications start from the main() function, which is the main.m file.
o AppDelegate.m: The AppDelegate receives messages from the application object during the lifetime of your application. The AppDelegate is called from the operating system, and contains events such as the didFinishLaunchingWithOptions, which is an event that iOS would be interested in knowing about.
o MainStoryBoard.storyboard: This is where the user interface is created. In past versions of xCode/iOS the user interface was stored within .xib (pronounced NIB) files. Although this method is still supported, Storyboards are a great improvement over .xib files for applications with complex navigation and many views.
o Supporting Files: The supporting files directory contains files such as the plist setting files (which contain customizable application settings), as well as string resource files that are used within your app.

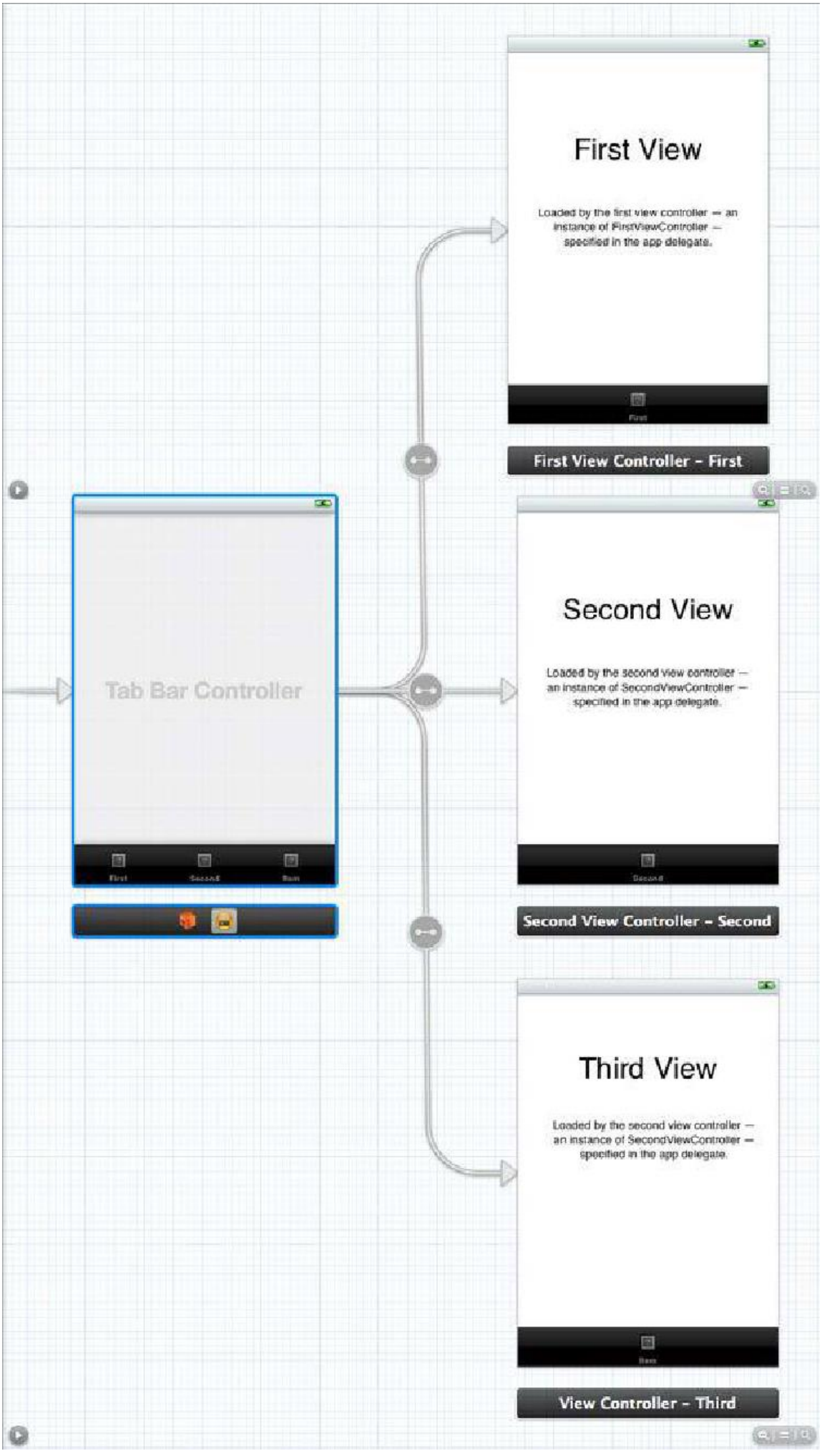
Getting to Know the xCode IDE

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

- Navigators: The left side of the xCode window is known as the navigator area. A variety of navigators enable you to list the contents of your project, find errors, search for code, and more. The remainder of this section introduces the Project Navigator, the Search Navigator,

and the Issue Navigator. Going from left to right, the project navigator is the first of the xCode navigators; the icon looks like a file folder. The Project Navigator simply shows the contents of your project or workspace, as shown in Figure 5.1. Double-clicking a file in the Project Navigator opens the file in a new window, and single-clicking opens the file within the xCode workspace.




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

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

9.b. Discuss the anatomy of IOS application

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.

10.a. Explain the various types of Notifications used in Windows Phone 7

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.

10.b. Write short note on the following with respect to Windows Phone

i)Offline Storage ii)Accelerometer

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

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 Reading Changed 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 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.

10.c. Explain GPS features for Windows Phone 7 Application

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

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