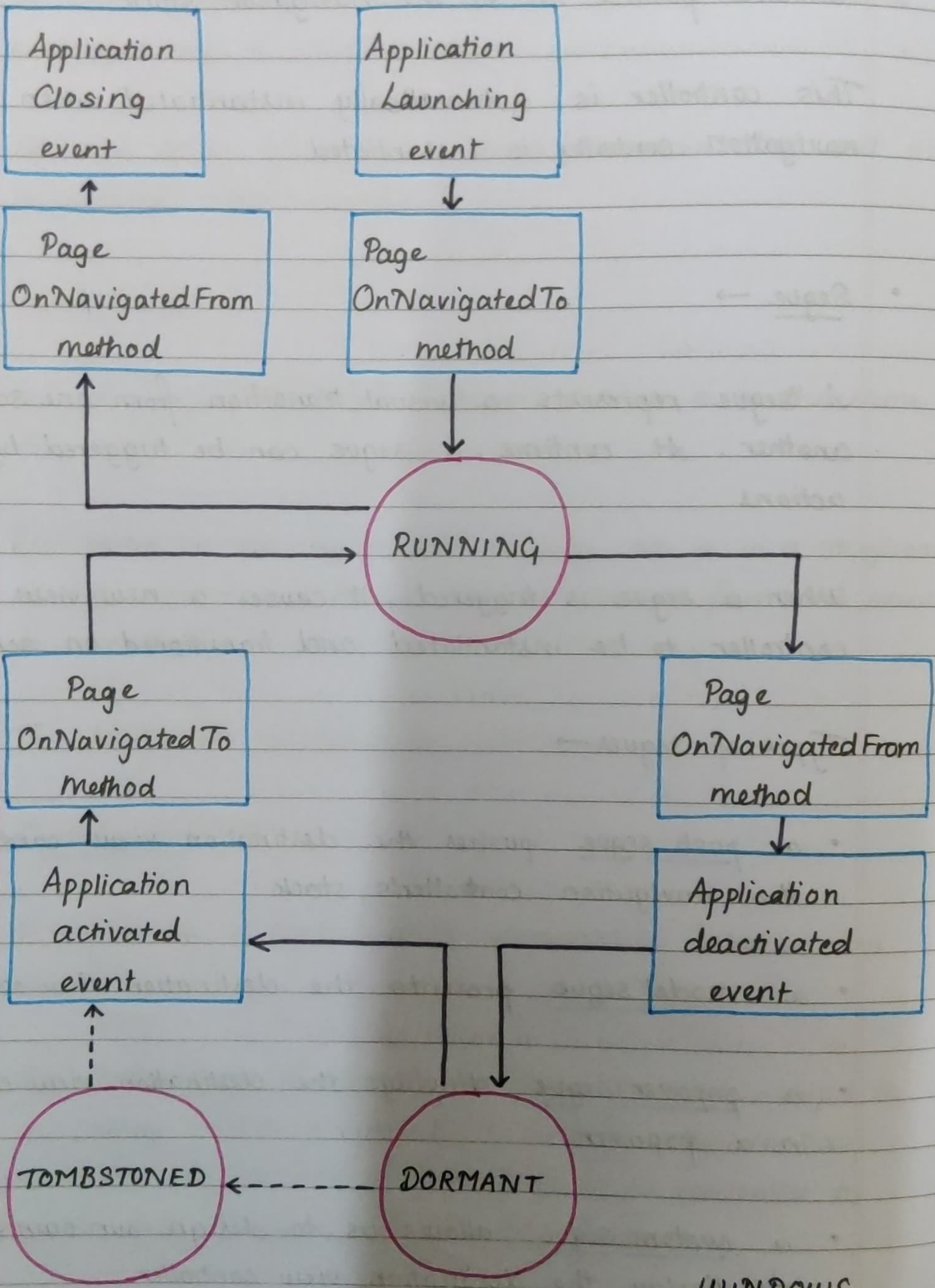
Unit - 5

Q1. Windows application life cycle


Application Closing event
Application Launching event
Page OnNavigatedFrom method
Page OnNavigatedTo method
RUNNING
Page OnNavigatedTo method
Page OnNavigatedFrom method
Application activated event
Application deactivated event
TOMBSTONED
DORMANT

WINDOWS APPLICATION LIFECYCLE

- The image illustrates the life cycle of a windows Phone application.

- In this diagram, the circles are application states

- The rectangles show either application or page level events where applications should manage their state.

- The Launching event →

→ The user can launch a new instance of your app, by selecting it from the installed applications list or from a Tile on Start, or by clicking on a toast notification associated with the app etc.

→ When your app is launched in this ~~wt~~ way, it should present a user interface that makes it clear that a new instance of the app was launched.

→ It is okay to provide the user context of their previous experience on the app (such as recently viewed documents), but it should not appear as if the user is returning to a previously running instance of the app.

→ When a new instance of your app is launched, ~~Aut~~ the Launching event is raised.

→ To ensure that your app loads quickly, you should execute as little code as possible in the handler for this event.

- Running →

→ After being launched, an app is Running

→ It continues to run until the user navigates forward, away from the app or backwards, past the app's first page

→ Windows phone apps provide a mechanism for users to quit or exit the app. Apps can also leave the running state when the phone's lock screen engages (unless if you have disabled application idle detection).

- OnNavigatedFrom method →

→ This method is called whenever the user navigates away from one of the pages in your app.

→ Whenever this method is called, your application should store the page state so that it can be restored if the user returns to the page and the page is no longer in memory.

→ Exception is in case of backward navigation, in which case there is no need to store the state because the page will be recreated the next time it is visited.

- The Deactivated Event →

→ The deactivated event is raised when the user navigates away from your app, by pressing start button or by ~~navigating to~~ launching another application.

→ This event is also raised if the device's lock screen is engaged, unless application idle detection is disabled.

→ In the handler for the deactivityed event, your application should save any unsaved application data so that it can be restored at a later time if necessary

→ It is possible for an event to be completely terminated after deactivated is called.

• Dormant →

→ After the deactivated event is raised, the operating system will attempt to put the app into a dormant state.

→ In this state, all of the application's threads are stopped and no processing takes place, but the application remains intact in memory.

→ If reactivated from dormant state, it doesnt need to do anything to re establish state, since it has been preserved.

→ If new apps are launched after an app is made dormant, and they require more memory than is available to provide a good user experience, then the operating system will begin to tombstone the dormant applications to free up memory.

- Tombstoned →

→ A tombstoned app has been terminated, but the os preserves info about it's navigation state and also preserves the state dictionaries the app populated during deactivated state.

→ The device will maintain tombstoning information for upto 5 apps at a time.

→ If a user navigates back to the application, it is relaunched and the preserved data can be used by the application to restore state.

- The Activated Event →

→ This is when a user navigates back to a dormant or tombstoned app

→ If IsApplicationInstancePreserved is true, then the app was dormant and state is automatically preserved by the operating system.
If it is false, then the app was tombstoned and should use state dictionary to restore the state.

- OnNavigatedTo Method →

→ This method is called when a user navigates to a page.

→ This includes when the app is first launched, when user navigates from page to page in the app, and when the app is relaunched after being made dormant or tombstoned.

→ In this method, your app should cheek if the page is a new instance. If it is not, then page state need not be restored, else the state dictionary should be used to restore the page state.

• The Closing Event →

→ This event is raised when the user navigates backwards past the first page of an app

→ In this case the app is terminated, and no state is saved.

→ In the closing event handler, your app can store the data that should persist ~~acc~~ across ~~all~~ instances.

→ There is a 10 second limit for the app to complete all the application and page navigation events.
If this limit is exceeded, then the app is terminated.

Q2. Events

- An event is a message sent by an object to signal the occurence of an action.

- The action could be caused by user interaction, such as touching the screen, or it could be triggered by the internal logic of a class.

- The object that raises the event is called the event sender.

- The object that ~~recieves~~ captures the event and responds to it is called the event reciever.

- Windows Phone events →

→ Generally speaking, windows phone events are CLR events, which can be ~~managed~~ handled with managed code.

→ Because the UI for a typical Windows phone based app is defined in markup (XAML), some of the principles of connecting UI events from markup elements to runtime code entity are similar to other web technologies such as ASP.NET or HTML DOM.

→ Generally you define the UI for your windows phone based app by generating XAML.

→ The XAML can be an output from a designer such as Blend or Visual Studio, or from a design surface in a larger IDE such as Windows Phone.

→ As a part of generating the XAML, you can wire event handlers for each individual UI elements when you define the other attributes for that element.

- **Defining event handler** →

→ Event handlers in the partial class are written as methods, based on the CLR delegates used by that particular event

→ The event handler methods can be public, or can have private access

→ The general recommendation is to not make your event handler methods public in the class.

- **Adding event handlers in managed code** →

→ XAML is not the only way to ~~add~~ assign an event handler to an object.

→ To add event handlers to any given object in the managed code, including to objects that are not even usable in XAML, you can use the CLR language specific syntax for adding event handlers.

- Routed events →

→ Windows Phone supports the concept of a routed event.

→ The following is a list of input events that are routed events →

- Key Down
- Key Up
- Got Focus
- Lost Focus
- MouseLeftButtonDown
- MouseLeftButtonUp
- MouseMove
- BindingValidationError

→ A routed event is an event that is potentially passed (routed) from a child object to each of it's successive parent objects in the object tree.

→ The object tree is approximated by the XAML structure of your UI, with the root of that tree being the root element in XAML.

→ The true object tree may vary somewhat from the XAML, because the object tree does not include XAML features such as property element tags.

- Handled property →

→ When you set the Handled property to true in event data, the routing stops for most handlers

→ The ~~routing~~ event does not continue along the route to notify other attached handlers of that particular event.

- User initiated events →

→ Windows Phone enforces that certain operations are only permitted in the context of a handler that handles user-initiated events.

eg. • Navigating from HyperLinkButton
• Accessing the primary clipboard API
• Windows phone user initiated events include mouse events (eg. MouseLeftButtonDown) and keyboard events (eg. KeyDown)

→ API calls that require user initiation should be called as soon as possible in an Event Handler.

- Removing event handlers →

→ In some circumstances, you may want to remove event handlers during ~~of~~ app lifetime.

→ For this you must use CLR language specific syntax.

Q3. Maps and Location

- The Maps App in a windows phone can show you where you are, where you want to go, and provide directions to get you there.

- It can also show you nearby shops or restaurants you might be interested in, and what other people are saying about them

- For displaying a map in your windows phone, use the Map control.

- To use the control, you have to select the ID_CAP_MAP capability in the app manifest file.

- The following code example shows how you can use XML to display a map control in your windows Phone 8 app

```
<!-- ContentPanel - place additional content here -->

<Grid x:Name = "Content Panel"  Grid.Row = "1"
                     Margin = "12, 0, 12, 0">

    <maps: Map />

</Grid>
```

- If you add the control ~~using~~ by writing XAML, you also have to add the ~~following~~ xmlns declaration to phone: PhoneApplicationPage element

- If you drag ~~the~~ and drop the Map control from the toolbox, this declaration is added automatically.

- Displaying a map with code (C#) -
The following code example shows how you can display a Map control in your Windows Phone 8 app

```
using Microsoft. Phone. Maps. Control;
    :
    :
 Map myMap = new Map ();

 ContentPanel. Children. Add (MyMap);
```

- Displaying a Map using Built-in Launcher →

The following ~~is~~ is a list of all the built-in launchers that display or manage maps :-

Mapstask	→	Launches the built-in Maps app and optionally marks a location.

Maps directions task → Launches the built in Maps app and displays directions.

MapDownloader task → Downloads maps for offline use

MapUpdater task → Checks for updates for offline maps that the user had previously downloaded.

- Specifying the center of the map (XAML) →

You can set the center of the map control by using the Center property, to which we assign a (latitude, longitude) pair

```
<!-- ContentPanel - place additional content here>

<Grid x: Name = "ContentPanel" Grid.Row = "1"
                Margin = "12, 0, 12, 0">

<maps: Map  x:Name= "MyMap"  Center = "47.61, 122.33"/>

</Grid>
```

- Specifying the zoom level of a map (XAML) →

This can be done using the ZoomLevel property to set the initial resolution to which you want to display the map

```
<!-- Content Panel - place additional content here>

<Grid x: Name = "ContentPanel", Grid.Row ="1"
                      Margin = "12, 0, 12, 0" >

<maps : Map  x: Name= "MyMap"  Center = "47.61, 122.33"
                      ZoomLevel = "10" />

</Grid>
```

- Displaying Landmarks and Pedestrian ~~et~~ features →

Set the LandmarksEnabled property to true to display Landmarks on the map control
Land marks are visible on the map only when ZoomLevel property is set to a value of 16 or Higher

Set the PedestrianFeaturesEnabled property to true to display pedestrian features such as public stairs
Pedestrian features are visible on the map only when the zoom level is set to 16 or higher.

```
<!- Content Panel - place additional content here>

<Grid  x: Name="ContentPanel"  Grid. Row= "1"
                      Margin= "12,0,12,0">

<maps: Map x:Name= "MyMap"  Center ="47.61, 122.33",
            ZoomLevel= "16", LandmarksEnabled ="true"
            PedestrianFeatures Enabled = "true" />

</Grid>
```