Lab assignment 6

Due date: Wednesday, June 15

40 points

Team of three to four students

Divide the team into two subgroups with two team members per group.

The group works on the lab assignment and can discuss the code with another group. Each group demonstrates the lab to each other. The team then upload the solution from one of the subgroups.

Write the contribution to the lab assignment, the contribution percentage (maximum of 100%) on the comment box for each team member.

All the team members must agree on the percentage. If the team members have 50% and the lab assignment grade is 40/40, they get only 20 points. The instructor will review the paper summarizing and validate the percentage of each team member. If you have problems with team members, please let me know.

Problem

In this lab assignment you demonstrate how to use and manage fragments. You build an app to play the hangman game.

Design

Defining and Adding a Fragment to an Activity Using a Layout XML File, Hangman, App Version 0

The Model is the `Hangman` class. It encapsulates a game where the user has to guess all the letters in a word but has only so many tries to do it. We keep our Model to the very minimum, and limit the number of possible words to just a few.

We provide a default value for the number of allowed guesses and an array of hard coded words
The instance variable `word` stores the word to guess.
The `guessesAllowed` and `guessesLeft` instance variables keep track of the number of guesses allowed and the number of guesses that the user still has. Finally, the `indexesGuessed` array keeps track of the indexes of the letters in the `String word` that the user has guessed correctly so far. A value of `true` means a correct guess at that index.

The Hangman constructor sets the value of the number of guesses allowed. It then randomly selects a word within the array words and instantiates the indexesGuessed array.

We provide methods to play the game , retrieve the state of completion of the word , and check if the game is over). We also provide accessors for guessesAllowed and guessesLeft.

The diagram below shows an actual game. At the bottom of the left pane, are the number of guesses that the user still has. Inside the upper right pane, are the letters that the user has correctly guessed so far, and we provide an `EditText` to input the next letter. At the bottom of the right pane, is the status or final result of the game.

Model

```
import java.util.Random;

public class Hangman {
    public static int DEFAULT_GUESSES = 6;
    private String [] words = { "ANDROID", "JAVA", "APP", "MOBILE" };
    private String word;
    private boolean [] indexesGuessed;
    private int guessesAllowed;
    private int guessesLeft;

    public Hangman( int guesses ) {
        if( guesses > 0 )
            guessesAllowed = guesses;
        else
            guessesAllowed = DEFAULT_GUESSES;
        guessesLeft = guessesAllowed;
        Random random = new Random( );
        int index = random.nextInt( words.length );
        word = words[index];
        indexesGuessed = new boolean[word.length( )];
    }

    public int getGuessesAllowed( ) {
        return guessesAllowed;
    }

    public int getGuessesLeft( ) {
        return guessesLeft;
    }

    public void guess( char c ) {
```

```
        boolean goodGuess = false;
        for( int i = 0; i < word.length( ); i++ ) {
            if( !indexesGuessed[i] && c == word.charAt( i ) ) {
                indexesGuessed[i] = true;
                goodGuess = true;
            }
        }
        if( !goodGuess )
            guessesLeft--;
    }

    public String currentIncompleteWord( ) {
        String guess = "";
        for( int i = 0; i < word.length( ); i++ )
            if( indexesGuessed[i] )
                guess += word.charAt( i ) + " ";
            else
                guess += "_ ";
        return guess;
    }

    public int gameOver( ) {
        boolean won = true;
        for( int i = 0; i < indexesGuessed.length; i++ )
            if( indexesGuessed[i] ==  false ) {
                won = false;
                break;
            }

        if( won ) // won
            return 1;
        else if( guessesLeft == 0 ) // lost
            return -1;
        else // game not over
            return 0;
    }
}
```

We provide a default value for the number of allowed guesses and an array of hard coded words. The instance variable word stores the word to guess. The guessesAllowed and guessesLeft instance variables keep track of the number of guesses allowed and the number of guesses that the user still has. Finally, the indexesGuessed array keeps track of the indexes of the letters in the String word that the user has guessed correctly so far. A value of true means a correct guess at that index.

The Hangman constructor sets the value of the number of guesses allowed. It then randomly selects a word within the array words, and instantiates the indexesGuessed array. We could envision a much more complex Model where another constructor could pull a list of possible words from a file, a database, or even a remote website, and then selects one at random.

We provide methods to play the game , retrieve the state of completion of the word, and check if the game is over. We also provide accessors for guessesAllowed and guessesLeft .

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="game_state_background">#F00F</color>
    <color name="game_result_background">#F0F0</color>
    <color name="game_control_background">#FF00</color>
    <color name="buttonColor">#F555</color>
    <color name="inputColor">#FFF0</color>
</resources>
```

AndroidManifest.xml

```
<activity android:name=".MainActivity"
    android:screenOrientation="landscape">
```

Activity_main.xml

There are two elements inside the top LinearLayout element: a fragment element, and a LinearLayout element, defined. We specify that this fragment is an instance of the GameControlFragment class, which we code later in this section. The top LinearLayout element's orientation is horizontal, so the fragment is positioned on the left side of the screen, and the LinearLayout on the right side of the screen. They both have the same android: layout_weight value of 1, so they both occupy 50% (= 1/(1 + 1)) of the screen (the fragment is in red, the LinearLayout in blue and green).
The android:layout_weight XML attribute of the LinearLayout. LayoutParams class enables us to specify how much space each View should occupy within its parent LinearLayout. We specify the value of 0dp for the android:layout_width attribute for both the fragment and LinearLayout elements. This is typical when the android_layout:weight attribute is used. The 0dp value means that we want to let the system decide how much space is allocated to the various elements based on the values for the layout_weight attributes.

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

    <LinearLayout tools:context="com.android.example.fragmenttest.MainActivity"
android:orientation="horizontal" android:layout_height="match_parent"
android:layout_width="match_parent" xmlns:tools="http://schemas.android.com/tools"
xmlns:android="http://schemas.android.com/apk/res/android">

    <fragment android:layout_height="match_parent" android:layout_width="0dp"
tools:layout="@layout/fragment_game_control" android:layout_weight="1"
android:name="com.android.example.fragmenttest.GameControlFragment"
android:id="@+id/gameControl"/>


    <LinearLayout android:orientation="vertical" android:layout_height="match_parent"
android:layout_width="0dp" android:layout_weight="1">

    <LinearLayout android:orientation="vertical" android:layout_height="0dp"
android:layout_width="match_parent" android:layout_weight="1"
        android:id="@+id/game_state"
        android:background="@color/game_state_background"/>

    <LinearLayout
        android:orientation="vertical"
        android:layout_height="0dp"
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:id="@+id/game_result"
        android:background="@color/game_result_background"
        android:gravity="center"/>

</LinearLayout>

</LinearLayout>
```

fragment_game_control.xml

We place the fragment_game_control.xml file in the layout directory. We use this layout XML file for the left pane of the screen, reserved by the fragment element. It is managed by a LinearLayout (and its background is red).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:background="#FF00"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
</LinearLayout>
```

GameControlFragment class

The `GameControlFragment` class only inflates the fragment_game_control.xml file, inside the `onCreateView` method. The first argument we pass to the `inflate` method is the resource id for the fragment, `fragment_game_control`. The second argument is the container that the

fragment goes into, the `ViewGroup` passed by the `onCreateView` method, `container`. The third argument is `false` because in this case, the inflated XML file is already inserted into `container`. Specifying `true` would create a redundant `ViewGroup` hierarchy.

```
public class GameControlFragment extends _________________ {
    public GameControlFragment( ) {
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState ) {
        return inflater.inflate( _______________,
                container, false );
    }
}
```

Run the app.

Adding GUI Components, Styles, Strings, and Colors, Hangman, App Version 1

In Version 1 of the app, we populate the left area of the screen (the fragment) with the two GUI components, and we start using the Model.

Updated activity_main.xml

In the updated activity_main.xml file, we give ids to the LinearLayout elements at the top right and bottom right of the screen so that we can access them later to place fragments in them. We change from hard coded colors to colors defined in the colors.xml file.

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

    <LinearLayout tools:context="com.android.example.hangman.MainActivity"
android:orientation="horizontal" android:layout_height="match_parent"
android:layout_width="match_parent" xmlns:tools="http://schemas.android.com/tools"
xmlns:android="http://schemas.android.com/apk/res/android">

    <fragment android:layout_height="match_parent" android:layout_width="0dp"
tools:layout="@layout/fragment_game_control" android:layout_weight="1"
android:name="com.android.example.hangman.GameControlFragment"
```

```
android:id="@+id/gameControl"/>


    <LinearLayout android:orientation="vertical" android:layout_height="match_parent"
android:layout_width="0dp" android:layout_weight="1">

    <LinearLayout android:id="@+id/game_state" android:orientation="vertical"
android:layout_height="0dp" android:layout_width="match_parent"
android:layout_weight="1" android:background="@color/game_state_background"/>
    <LinearLayout android:id="@+id/game_result" android:orientation="vertical"
android:layout_height="0dp" android:layout_width="match_parent"
android:layout_weight="1" android:background="@color/game_result_background"/>

</LinearLayout>

</LinearLayout>
```

Colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>

    <color name="colorPrimary">#3F51B5</color>

    <color name="colorPrimaryDark">#303F9F</color>

    <color name="colorAccent">#FF4081</color>

    <color name="buttonColor">#F555</color>

    <color name="game_control_background">#FF00</color>

    <color name="game_state_background">#F00F</color>

    <color name="game_result_background">#F0F0</color>

    <color name="inputColor">#FFF0</color>

</resources>
```

Updated fragment_game_control.xml

In the updated fragment_game_control.xml file, we place two Linear-Layouts in it: they each contain a basic GUI component (a button and a label). By placing one component per layout manager and giving

each layout manager the same android: layout_weight value of 1 via the wrapCenterWeight1 style, we space out the components evenly. The wrapCenterWeight1 style, which also sets android:layout_width and android:layout_height to wrap_content and android:gravity to center, is defined in the styles.xml file. The use of styles simplifies the coding of our layout XML files.

```
<?xml version="1.0" encoding="utf-8"?>
   <LinearLayout android:gravity="center"
android:background="@color/game_control_background" android:orientation="vertical"
android:layout_height="match_parent" android:layout_width="match_parent"
xmlns:android="http://schemas.android.com/apk/res/android">

    <LinearLayout style="_____________________">

    <________ style="_________________" android:onClick="play"
android:text="@string/play"/>

</LinearLayout>

    <LinearLayout style="_____________________________">

    <___________ style="@style/textStyle" android:id="@+id/status"/>

</LinearLayout>

</LinearLayout>
```

styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="wrapCenterWeight1">

    <item name="android:layout_height">wrap_content</item>

    <item name="android:layout_width">wrap_content</item>

    <item name="android:gravity">center</item>

    <item name="android:layout_weight">1</item>

</style>


    <style parent="@android:style/TextAppearance" name="textStyle">

    <item name="android:layout_width">wrap_content</item>

    <item name="android:layout_height">wrap_content</item>

    <item name="android:textSize">36sp</item>

    <item name="android:gravity">center</item>
```

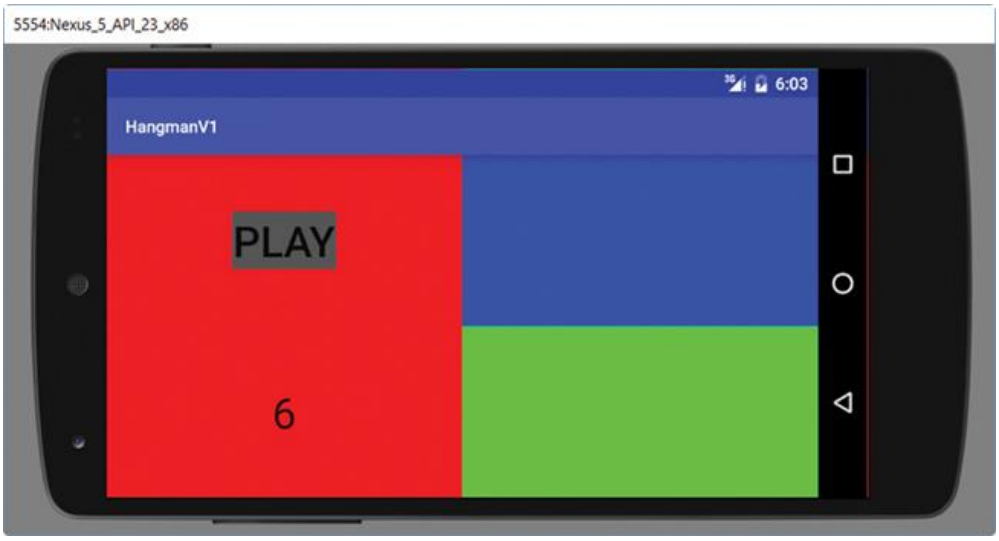
```
</style>

    <style parent="textStyle" name="buttonStyle">

    <item name="android:background">@color/buttonColor</item>

</style>
</resources>
```

MainActivity class

The MainActivity class includes a Hangman instance variable representing the Model. Since the onCreate method can be called several times during the life cycle of this activity, we only instantiate it if it has not been instantiated before (i.e., if it is null). We call the getGuessesLeft method of the Hangman class with it in order to set the text inside the TextView of the fragment. We retrieve that TextView using the findViewById method. The class also includes the play method, which is implemented as a do-nothing method at this point (if we do not implement it, if we run and click on the button, the app will stop working).

```
public class MainActivity extends AppCompatActivity {
     private _________________ game;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if ( game == _________________ )
            game = new _____________________________;
        setContentView( R.layout.activity_main );
        TextView status = ______________________________________;
        status.setText( "" _____________________);
    }

    public void play( View view ) {
    }
    }
```

Run the app.



Defining a Fragment Using a Layout XML File and Adding the Fragment to an Activity by Code, Hangman, App Version 2

In Version 2, we display the fragment on the top right pane (with the blue background) of the screen showing the state of completion of the word, and an `EditText`for the user to enter a letter. We code the fragment in an XML file, fragment_game_state.xml, and create the fragment by code.

Furthermore, in order to better focus on fragments and keep things simple, we run the app in horizontal orientation only by adding this line to the AndroidManifest.xml file inside the `activity` element:

android:screenOrientation=”landscape”

styles.xml

```
<style parent="textStyle" name="editTextStyle">

    <item name="android:inputType">textCapCharacters</item>

    <item name="android:background">@color/inputColor</item>

    <item name="android:hint">Type a letter here</item>

</style>
```

Fragment_game_state.xml

The fragment_game_state.xml file includes a TextView, which shows the state of completion of the word and an EditText, where the user can enter a letter. They are styled with textSyle and editTextStyle, which are both defined in the styles.xml file. For the left pane fragment, we wrap each element around a LinearLayout element styled with the wrapCenterWeight1 style, so that the elements are centered and spaced evenly.

We want to close the soft keyboard when the user touches the Done key. In order to do that, we set the value of the android:imeOptions attribute, inherited from TextView, to actionDone . IME stands for Input Method Editor, and it enables the user to enter text. It is not restricted to a keyboard—the user can also enter text by voice. The attribute android:imeOptions enables us to specify the options that we want to enable.

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

    <LinearLayout android:gravity="center" android:orientation="vertical"
android:layout_height="match_parent" android:layout_width="match_parent"
xmlns:tools="http://schemas.android.com/tools"
xmlns:android="http://schemas.android.com/apk/res/android">


    <LinearLayout style="_____________________">

    <_____________style=”_____________" android:id="@+id/state_of_game"/>

</LinearLayout>


    <LinearLayout style="_____________________">

    <___________ style="_____________" android:id="@+id/letter"
android:imeOptions="_____________"/>

</LinearLayout>

</LinearLayout>
```

We need to show the game state fragment when the app starts. We first need to create the fragment and add it to the activity, then set the text inside the `TextView` element of the fragment. For that, we can call the `currentIncompleteWord` method of our Model. To incorporate a fragment into an activity, we need to implement the following steps:

- Get a reference to the fragment manager for this `Activity`.
- Create a fragment transaction.
- Create a fragment.
- Have the fragment transaction add the fragment to a container (a `ViewGroup`).
- Commit the fragment transaction.

Method Description

FragmentTransaction beginTransaction( ) Starts a series of operations for this fragment manager. Returns a FragmentTransaction.

Fragment findFragmentById(int id) Returns a Fragment identified by its id.

Fragment findFragmentByTag(String tag)

Selected methods of the FragmentTransaction class

Method Description

FragmentTransaction add(int containerViewId, Fragment fragment, String tag) Adds a fragment to an activity. ContainerViewId is the resource id of the container that the fragment will be placed in; fragment is the fragment to be added. Tag is an optional tag name for the fragment. The added fragment can later be retrieved by the fragment manager using the findFragmentByTag method. Returns the FragmentTransaction reference calling this method; thus, method calls can be chained.

FragmentTransaction add(int containerViewId, Fragment fragment) Calls the method above with a null tag parameter.

FragmentTransaction add(Fragment fragment, String tag) Calls the method above with a resource id parameter equal to 0.

FragmentTransaction hide(Fragment fragment) Hides fragment (i.e., hides its View).

FragmentTransaction show(Fragment fragment) Shows fragment (i.e., shows its View if it had been hidden before—the default is that the fragment is shown).

FragmentTransaction remove(Fragment fragment) Remove fragment from its activity.

FragmentTransaction replace(int containerViewId, Fragment fragment, String tag) Similar to the first add method, but instead replaces an existing fragment inside the container whose id is containerViewId.

FragmentTransaction replace(int containerViewId, Fragment fragment) Calls the method above with a null tag parameter.

int commit( ) Schedules that fragment transaction to be committed.

FragmentTransaction addToBackStack(String name) Adds this transaction to the back stack of the activity; name is optional and stores the name of that back stack state.

Updated MainActivity class

When the app starts, the onCreate method executes, and we create the fragment and attach it to the activity. We get a reference to the fragment manager for this activity by calling the getFragmentManager method from the Activity class. We only want to create a new GameStateFragment and add it to the activity if it has not been created yet. Using the findFragmentById method, we check if there is already a fragment with the id game_state within this activity. If there is, we do nothing. If there is not, we create one and add it to the activity ..

```
public class MainActivity extends AppCompatActivity {

    private Hangman game;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (game == _____)
            game = new Hangman(_____________________________);
        setContentView(R.layout.activity_main);
        TextView status = (TextView) findViewById(_________________);
        status.setText("" + _____________________);
        if (getSupportFragmentManager().findFragmentById(__________) == null) {
            getSupportFragmentManager()._______
                    .setReorderingAllowed(true)
                    .add(___________,_____________________, null)
                    .______________;
        }
public Hangman getGame()
{ return game;}
```

GameStateFragment class

The GameStateFragment class inherits from Fragment, we provide a mandatory default constructor . We inflate the fragment_game_state.xml file inside the onCreateView method. When the fragment transaction is committed, the constructor, the onAttach, onCreate, onCreateView, onActivityCreated, and onStart methods of the Fragment life cycle are called in that order. When the onStart method executes, the fragment's View and the TextView inside it have been instantiated. Thus, we can set the text inside the TextView at that point.

We get a reference to the fragment's View using the getView method inherited from the Fragment class. With it, we can call the findViewById method to access any View inside it that has been given an id. We call it and assign the TextView whose id is state_of_game to gameStateTV. We get a reference to the parent Activity of this fragment, casting it to a MainActivity. We chain method calls to getGame and currentIncompleteWord in order to retrieve the game's incomplete word and assign the resulting value to the text of gameStateTV.

The getActivity and getView methods of the Fragment class

Method	Description
Activity getActivity( )	Returns the activity that this fragment belongs to.
View getView( )	Returns the root View for this fragment.

```
public class GameStateFragment extends Fragment {

    public GameStateFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_game_state,
                container, false);
    }
    public void onStart( ) {
        super.onStart( );
        View fragmentView = _________________:
        TextView gameStateTV
                = ( __________ ) fragmentView.findViewById( _____________ );
        MainActivity fragmentActivity = ( MainActivity ) ____________;
        gameStateTV.setText( fragmentActivity._____________;    }

}
```

Defining and Adding a Fragment to an Activity by Code, Hangman, App Version 3

In Version 3, we display the fragment on the bottom right pane (with the green background) of the screen showing a message about the result of the game in a `TextView`. This time, we do not use an XML file to define the fragment, we define and create it entirely by code.

Defining the fragment's Graphical User Interface (GUI) is similar to defining a View by code instead of using an XML file. We do this in the `GameResultFragment` class. Creating the fragment is identical to creating a game state fragment. It takes place in the `MainActivity` class.

Updated MainActivity class

```
if (getSupportFragmentManager().findFragmentById(R.id.game_result) == null) {
    getSupportFragmentManager().beginTransaction()
            .setReorderingAllowed(true)
            .add(_____________________________________)
            .commit();
```

GameResultFragment class

In the GameResultFragment class, we define the fragment by code. This fragment only has a TextView in it. Furthermore, we need to access that TextView to set its text when the game ends. Rather than giving the TextView an id when we create it, we declare it as an instance variable so that we have a direct reference to it inside the GameResultFragment The onCreateView method calls the setUpFragmentGui method in order to create the GUI for this fragment. It returns the View returned by its super method. Since it is possible that onCreateView is called several times during the life cycle of the fragment, we can in turn expect setUpFragmentGui to be called several times too. Thus, we want to instantiate gameResultTV and add it to the ViewGroup that is the root View of the fragment only once, when it is null. We center the text inside gameResultTV.

In the onStart method which is automatically called after onCreateView and onActivityCreated, we hard code the text inside gameResultTV to "GOOD LUCK" .

We need to center the TextView of GameResultFragment within its parent LinearLayout. Inside the activity_main.xml file, we add the following attribute-value pair to the last LinearLayout element (the one whose id is game_result and is located in the right bottom pane):

```
public class GameResultFragment extends Fragment {
    private TextView gameResultTV;
    private float ourFontsize = 50f;

    @Override
    public View onCreateView( LayoutInflater inflater,
                              ViewGroup container, Bundle savedInstanceState ) {
        setUpFragmentGui( container );
        return super.onCreateView( inflater, container,
                savedInstanceState ) ;
```

```
    }

    public void setUpFragmentGui( ViewGroup container ) {
        if( gameResultTV == null ) {
            gameResultTV = new TextView( _______________ );
            gameResultTV.setGravity( Gravity.CENTER );
            gameResultTV.setTextSize(TypedValue.COMPLEX_UNIT_SP, ourFontsize);
            container.addView( ____________________ );
        }
    }

    public void onStart( ) {
        super.onStart( );
        ____________( "GOOD LUCK" );
    }
    public void setResult( String result ) {
        ____________________( result );
    }
}
```

Run the app. All three fragments are now present:

. The left pane (game control) fragment in red is defined and created using XML.

. The top right pane (game status) fragment in blue is defined in XML and created by code.

. The bottom right pane (game result) fragment in green is defined and created by code.

Communication between Fragments and Their Activity: Enabling Play, Hangman, App Version 4

We have now created an activity with three fragments inside, defined and created in different ways. In Version 4, we complete the app by processing the user's letter and enabling the user to play the game. We also illustrate how fragments can communicate with their activity when processing an event.

The user enters a letter in the game state fragment, then clicks on the PLAY button on the left pane, which triggers execution of the `play` method inside the `MainActivity` class. To enable play, we do the following:

- Capture the letter entered by the user.
- Call the `guess` method of `Hangman`.
- Update the number of guesses left in the `TextView` inside the left pane.
- Update the incomplete word in the `TextView` inside the top right pane.
- Clear the `EditText`.
- If the game is over, update the message in the `TextView` inside the bottom right pane and clear the hint in the `EditText`.

play method

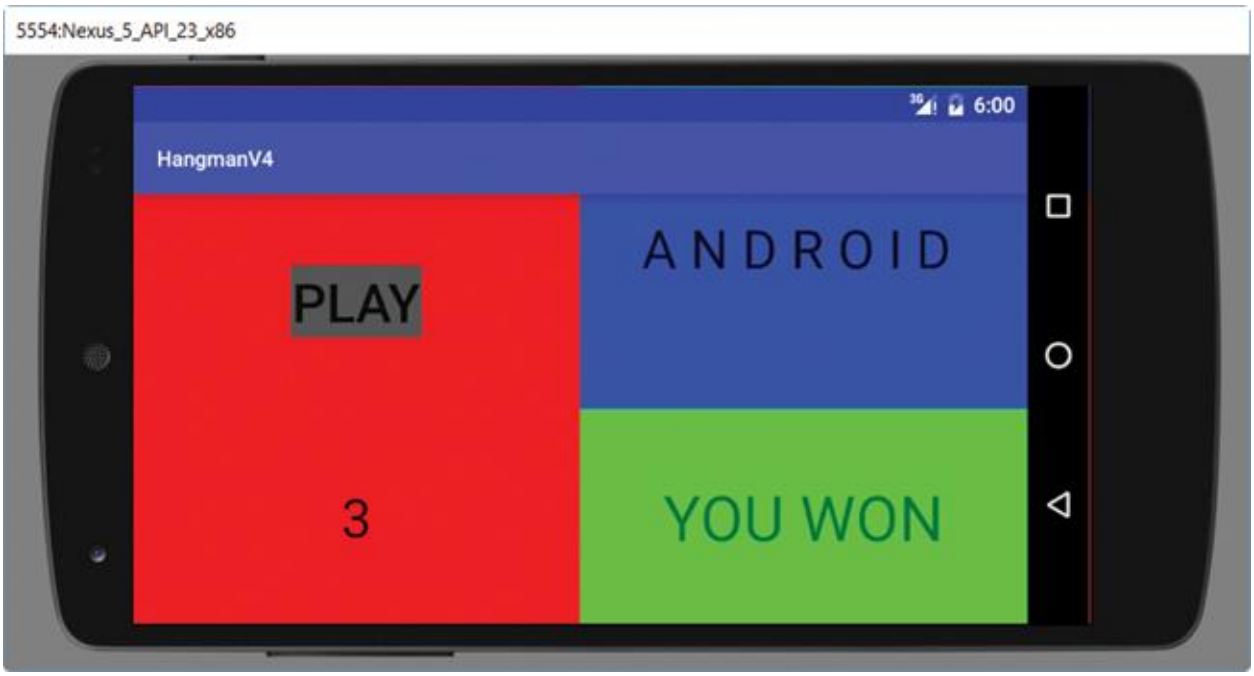
```
public void play(View view) {
    EditText input = (EditText) findViewById(R.id.letter);
    Editable userText = input.getText();
    if (userText != null && userText.length() > 0) {
        // update number of guesses left
        char letter = userText.charAt(0);
        game.guess(letter);
        TextView status = (TextView) findViewById(R.id.status);
        status.setText("" + _____________________);

        // update incomplete word

        GameStateFragment gsFragment = (GameStateFragment)
getSupportFragmentManager().findFragmentById(_____________________);
        View gsFragmentView = gsFragment._____________________;
        TextView gameStateTV = (TextView)
                gsFragmentView.findViewById(_____________________);
        gameStateTV.setText(_____________________);

        // clear EditText
        input.setText("");
        // check if there is only one guess left
        if( game.getGuessesLeft( ) == 1 ) {
            BackgroundFragment background = ( BackgroundFragment )
                    getSupportFragmentManager().findFragmentByTag( __________ );
            GameResultFragment grFragment = ( GameResultFragment )
                    getSupportFragmentManager().findFragmentById( _____________ );
            // retrieve warning and display it
```

```
            grFragment.setResult( _____________________ );
        }

        int result = game.gameOver();
        if (result != 0) /* game is over */ {
            GameResultFragment grFragment = (GameResultFragment)
                    getSupportFragmentManager().findFragmentById(_____________);

            // update TextView in result fragment
            if (result == 1)
                _______________("YOU WON");
            else if (result == -1)
                ____________("YOU LOST");

            // delete hint in EditText
            input.setHint("");
        }
    }
}
```

setResult method of the GameResultFragment class

```
public void setResult( String result ) {
    _________________.setText( result );
}
```

Run the app.



Using an Invisible Fragment, Hangman, App Version 5

We can also use a fragment that can perform some work in the background of the app, without any visual representation. For this, we use the `add` method of the `FragmentTransaction` class that does not have a `View` id parameter. If we want to be able to retrieve the fragment later, we should provide a tag name. Such a fragment could perform some work in the background of the app while the user is allowed to interact with the app. It could, for example, retrieve some data from a file, a database, a remote URL, or possibly use the device's GPS to retrieve some live location data.

BackgroundFragment class

```
public class BackgroundFragment extends Fragment {
    public BackgroundFragment( ) {
    }

    public String warning( ) {
        return "ONLY 1 LEFT!";
    }
}
```

Updated MainActivity class

```
if( getSupportFragmentManager().findFragmentByTag( ___________ ) == null ) {
    getSupportFragmentManager().beginTransaction( )
            .setReorderingAllowed(true)
            .add(_________________)
            .commit();
}
//check if there is only one guest left
//Your code
```

Run the app. Check for the number of guess left.

Grading:

1. Submit the project in zip file (extension zip)
2. A pdf document that contains the content of the following files: GameControlFragment class, GameResultFragment class, BackgroundFragment class, MainActivity class, and AndroidManifest.xml.
3. A zoom link or YouTube that demonstrate the user of the Hangman app (less than 5 minutes). Write the link on the comment section in the dropbox.
   Your video should demonstrate the following features of the app:
   a. Player won
   b. Player lost
   c. Number of guest left

Note:
App is running correctly and met all the requirements is 30 points, but
Incomplete item 1(-30 points)
Incomplete item 2 (-15 points)
Incomplete item 3 (-15 points)

Processing the Keyboard Input Directly, Hangman, App Version 6

In Version 6, we enable play as soon as the user closes the keyboard. We can either eliminate or convert the PLAY button to a “Play Another Game” button

The EditText is located inside the game state fragment, so event handling for Version 6 takes place in that fragment. OnEditorActionListener, a public static inner interface of the TextView class, provides the functionality to handle a key event inside an EditText.

The updated GameStateFragment class defines the private class OnEditorHandler, which implements the TextView.OnEditorActionListener interface. It overrides its only method, onEditorAction. It does the following:

- Hides the keyboard
- Plays

Since we already have a play method in the MainActivity class, we can reuse its code. The existing play method accepts a View parameter, expected to be a Button, which the method does not actually use. We could call it and pass null, but it is not good practice to pass null to a method from another class, since in theory (and in general), it could use that parameter to call methods. Thus, we create an additional play method in the MainActivity class that accepts no parameter and does exactly the same thing as the existing play method (which we can leave as a do-nothing method—if we choose to delete it, we also need to delete the android: onClick attribute in the fragment_game_control.xml file).

In order to close a keyboard by code, we can use an InputMethodManager reference. The InputMethodManager class manages the interaction between an application and its current input method. That class does not have a constructor but we can obtain an InputMethodManager reference by calling the getSystemService of the Context class using an Activity reference (which is a Context reference since Activity inherits from Context).

The getSystemService method has the following API:

Object getSystemService(String nameOfService)

Its String parameter represents a service.

Depending on the service specified, it returns a manager object reference for that service. For example, if we specify LOCATION_SERVICE, it returns a LocationManager reference that we can use to gather location data from the device's GPS system. If we specify WIFI_SERVICE, it returns a WifiManager reference that we can use to access data regarding the device's Wi-Fi connectivity. Because that method returns a generic Object, we need to typecast the returned object reference to what we expect. We want to obtain an InputMethodManager reference, so we pass the argument Context.INPUT_METHOD_SERVICE and cast the resulting object reference to an InputMethodManager as follows

InputMethodManager inputManager = (InputMethodManager)

getActivity( ).getSystemService(Context.INPUT_METHOD_SERVICE);

To close the keyboard, we use one of the hideSoftInputFromWindow methods of the InputMethodManager class.

The first parameter has type IBinder, an interface that encapsulates a remotable object. A remotable object is an object that can be accessed outside its context via a proxy. We want to pass to hideSoftInputFromWindow an IBinder reference that represents the window that the current View (in this case the EditText) is attached to. We call the getCurrentFocus method of the Activity class with the current Activity object. It returns the View that has the focus in the current window. Using that View, we call the get WindowToken of the View class. It returns a unique token that identifies the window that this View is attached to. At line 86, we chain these two method calls using the expression:

getActivity( ).getCurrentFocus( ).getWindowToken( )

The two constants of the InputMethodManager class that can be used as values for the second parameter of the hideSoftInputFromWindow method. Since the user explicitly opens the keyboard when trying to enter some input, we should not use the HIDE_IMPLICIT_ONLY constant in this case. We use HIDE_NOT_ALWAYS, so the second argument of we method call is:

InputMethodManager.HIDE_NOT_ALWAYS

When we run the app, play happens as soon as the keyboard closes. We no longer need to click on the PLAY button.

Updated MainActivity

```
public void play() {
    Code the same as play(View view) in version 5.
}

public void play( View view ) {
}
```

Updated GameStateFragment class

```
public void onStart( ) {

    gameStateTV.setText( fragmentActivity.getGame( )
            .currentIncompleteWord( ) );
    EditText answerET
            = (EditText) fragmentView.findViewById( R.id.letter );
____________________ editorHandler = _____________________________;
    answerET._____________________________________________________;


}
```

```
private class OnEditorHandler implements ____________________________ {
    public boolean onEditorAction(TextView v,
                                  int keyCode, KeyEvent event) {
        // hide the keyboard
        InputMethodManager inputManager = (InputMethodManager)
                ____________________.getSystemService(_____________________________);
        inputManager.hideSoftInputFromWindow(
                getActivity().getCurrentFocus().getWindowToken(),
                ____________________________________________);

        // play
        MainActivity fragmentActivity = _____________________ getActivity();
        ____________________________________________

        return true;
    }
}
```

Grading:

1. Submit the project in zip file (extension zip)
2. A pdf document that contains the content of the following files: GameStateFragment class, and MainActivity class.
3. A zoom link or YouTube that demonstrate the user of the Hangman app (less than 5 minutes). Write the link on the comment section in the dropbox.
   Your video should demonstrate the following features of the app in version 6:
   a. Player won
   b. Player lost
   c. Number of guest left

Note:
App is running correctly and met all the requirements is 10 points, but
Incomplete item 1(-10 points)
Incomplete item 2 (-3 points)
Incomplete item 3 (-3 points)