MAD ASSIGNMENT-3

G YASHWANTH

2311cs020259

ACTIVITY\_MAIN.XML

<?xml version="1.0" encoding="utf-8"?>  
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:padding="16dp">  
  
 <LinearLayout  
 android:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <EditText  
 android:id="@+id/firstName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="First Name" />  
  
 <EditText  
 android:id="@+id/lastName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Last Name" />  
  
 <EditText  
 android:id="@+id/address"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Address Line" />  
  
 <Spinner  
 android:id="@+id/countrySpinner"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content" />  
  
 <TextView  
 android:id="@+id/dobText"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Date of Birth" />  
  
 <Button  
 android:id="@+id/dobButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Select Date of Birth" />  
  
 <TextView  
 android:id="@+id/timeText"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Preferred Time" />  
  
 <Button  
 android:id="@+id/timeButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Select Time" />  
  
 <EditText  
 android:id="@+id/cardName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Name on Card" />  
  
 <EditText  
 android:id="@+id/cardNumber"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Card Number" />  
  
 <EditText  
 android:id="@+id/expiryDate"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Expiry Date" />  
  
 <EditText  
 android:id="@+id/cvv"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="CVV" />  
  
 <RadioGroup  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <RadioButton  
 android:id="@+id/payByCard"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Pay by Card" />  
  
 <RadioButton  
 android:id="@+id/payByTransfer"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Pay by Transfer" />  
 </RadioGroup>  
  
 <CheckBox  
 android:id="@+id/saveInfo"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Save the information for later" />  
  
 <Button  
 android:id="@+id/registerButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Register" />  
 </LinearLayout>  
</ScrollView>

MAINACTIVITY.JAVA

package com.example.registratioform259;  
  
import android.app.DatePickerDialog;  
import android.app.TimePickerDialog;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.\*;  
import androidx.appcompat.app.AppCompatActivity;  
import java.util.Calendar;  
  
public class MainActivity extends AppCompatActivity {  
 private EditText firstName, lastName, address, cardName, cardNumber, expiryDate, cvv;  
 private Spinner countrySpinner;  
 private RadioButton payByCard, payByTransfer;  
 private CheckBox saveInfo;  
 private Button dobButton, timeButton, registerButton;  
 private TextView dobText, timeText;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 firstName = findViewById(R.id.*firstName*);  
 lastName = findViewById(R.id.*lastName*);  
 address = findViewById(R.id.*address*);  
 cardName = findViewById(R.id.*cardName*);  
 cardNumber = findViewById(R.id.*cardNumber*);  
 expiryDate = findViewById(R.id.*expiryDate*);  
 cvv = findViewById(R.id.*cvv*);  
 countrySpinner = findViewById(R.id.*countrySpinner*);  
 payByCard = findViewById(R.id.*payByCard*);  
 payByTransfer = findViewById(R.id.*payByTransfer*);  
 saveInfo = findViewById(R.id.*saveInfo*);  
 dobButton = findViewById(R.id.*dobButton*);  
 timeButton = findViewById(R.id.*timeButton*);  
 registerButton = findViewById(R.id.*registerButton*);  
 dobText = findViewById(R.id.*dobText*);  
 timeText = findViewById(R.id.*timeText*);  
  
 // Populate country spinner  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(this,  
 R.array.*countries*, android.R.layout.*simple\_spinner\_item*);  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 countrySpinner.setAdapter(adapter);  
  
 dobButton.setOnClickListener(v -> showDatePicker());  
 timeButton.setOnClickListener(v -> showTimePicker());  
  
 registerButton.setOnClickListener(v -> {  
 if (validateInputs()) {  
 Toast.*makeText*(this, "Registration Successful", Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
 }  
  
 private void showDatePicker() {  
 Calendar calendar = Calendar.*getInstance*();  
 DatePickerDialog datePicker = new DatePickerDialog(this,  
 (view, year, month, dayOfMonth) -> dobText.setText(dayOfMonth + "/" + (month + 1) + "/" + year),  
 calendar.get(Calendar.*YEAR*), calendar.get(Calendar.*MONTH*), calendar.get(Calendar.*DAY\_OF\_MONTH*));  
 datePicker.show();  
 }  
  
 private void showTimePicker() {  
 Calendar calendar = Calendar.*getInstance*();  
 TimePickerDialog timePicker = new TimePickerDialog(this,  
 (view, hourOfDay, minute) -> timeText.setText(String.*format*("%02d:%02d", hourOfDay, minute)),  
 calendar.get(Calendar.*HOUR\_OF\_DAY*), calendar.get(Calendar.*MINUTE*), true);  
 timePicker.show();  
 }  
  
 private boolean validateInputs() {  
 if (firstName.getText().toString().isEmpty() || lastName.getText().toString().isEmpty() ||  
 address.getText().toString().isEmpty() || cardName.getText().toString().isEmpty() ||  
 cardNumber.getText().toString().isEmpty() || expiryDate.getText().toString().isEmpty() ||  
 cvv.getText().toString().isEmpty() || dobText.getText().toString().isEmpty() ||  
 timeText.getText().toString().isEmpty()) {  
 Toast.*makeText*(this, "All fields are required", Toast.*LENGTH\_SHORT*).show();  
 return false;  
 }  
 return true;  
 }  
}

STRINGS.XML

<resources>  
 <string name="app\_name">259RegistrationForm</string>  
 <string-array name="countries">  
 <item>USA</item>  
 <item>Canada</item>  
 <item>UK</item>  
 <item>India</item>  
 <item>Australia</item>  
 </string-array>  
</resources>

FRAGMENTACTIVITY.JAVA

package androidx.fragment.app;  
  
import static androidx.activity.result.contract.ActivityResultContracts.StartIntentSenderForResult;  
  
import android.annotation.SuppressLint;  
import android.app.Activity;  
import android.content.Context;  
import android.content.Intent;  
import android.content.IntentSender;  
import android.content.res.Configuration;  
import android.os.Bundle;  
import android.os.Handler;  
import android.util.AttributeSet;  
import android.view.LayoutInflater;  
import android.view.MenuItem;  
import android.view.View;  
import android.view.Window;  
  
import androidx.activity.ComponentActivity;  
import androidx.activity.OnBackPressedDispatcher;  
import androidx.activity.OnBackPressedDispatcherOwner;  
import androidx.activity.result.ActivityResultCallback;  
import androidx.activity.result.ActivityResultRegistry;  
import androidx.activity.result.ActivityResultRegistryOwner;  
import androidx.activity.result.contract.ActivityResultContract;  
import androidx.annotation.CallSuper;  
import androidx.annotation.ContentView;  
import androidx.annotation.LayoutRes;  
import androidx.annotation.MainThread;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.core.app.ActivityCompat;  
import androidx.core.app.MultiWindowModeChangedInfo;  
import androidx.core.app.OnMultiWindowModeChangedProvider;  
import androidx.core.app.OnPictureInPictureModeChangedProvider;  
import androidx.core.app.PictureInPictureModeChangedInfo;  
import androidx.core.app.SharedElementCallback;  
import androidx.core.content.OnConfigurationChangedProvider;  
import androidx.core.content.OnTrimMemoryProvider;  
import androidx.core.util.Consumer;  
import androidx.core.view.MenuHost;  
import androidx.core.view.MenuProvider;  
import androidx.lifecycle.Lifecycle;  
import androidx.lifecycle.LifecycleOwner;  
import androidx.lifecycle.LifecycleRegistry;  
import androidx.lifecycle.ViewModelStore;  
import androidx.lifecycle.ViewModelStoreOwner;  
import androidx.loader.app.LoaderManager;  
import androidx.savedstate.SavedStateRegistry;  
import androidx.savedstate.SavedStateRegistryOwner;  
  
import java.io.FileDescriptor;  
import java.io.PrintWriter;  
import java.util.Collection;  
  
public class FragmentActivity extends ComponentActivity implements  
 ActivityCompat.OnRequestPermissionsResultCallback,  
 ActivityCompat.RequestPermissionsRequestCodeValidator {  
  
 static final String *LIFECYCLE\_TAG* = "android:support:lifecycle";  
  
 final FragmentController mFragments = FragmentController.*createController*(new HostCallbacks());  
final LifecycleRegistry mFragmentLifecycleRegistry = new LifecycleRegistry(this);  
  
 boolean mCreated;  
 boolean mResumed;  
 boolean mStopped = true;

public FragmentActivity() {  
 super();  
 init();  
 }  
  
@ContentView  
 public FragmentActivity(@LayoutRes int contentLayoutId) {  
 super(contentLayoutId);  
 init();  
 }  
  
 private void init() {  
 getSavedStateRegistry().registerSavedStateProvider(*LIFECYCLE\_TAG*, () -> {  
 markFragmentsCreated();  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_STOP*);  
 return new Bundle();  
 });  
 addOnConfigurationChangedListener(newConfig -> mFragments.noteStateNotSaved());  
   
 addOnNewIntentListener(newConfig -> mFragments.noteStateNotSaved());  
 addOnContextAvailableListener(context -> mFragments.attachHost(null /\*parent\*/));  
 }  
  
 @SuppressWarnings("deprecation")  
 @Override  
 @CallSuper  
 protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  
 mFragments.noteStateNotSaved();  
 super.onActivityResult(requestCode, resultCode, data);  
 }  
  
  
public void supportFinishAfterTransition() {  
 ActivityCompat.*finishAfterTransition*(this);  
 }  
  
public void setEnterSharedElementCallback(@Nullable SharedElementCallback callback) {  
 ActivityCompat.*setEnterSharedElementCallback*(this, callback);  
 }  
  
public void setExitSharedElementCallback(@Nullable SharedElementCallback listener) {  
 ActivityCompat.*setExitSharedElementCallback*(this, listener);  
 }  
  
public void supportPostponeEnterTransition() {  
 ActivityCompat.*postponeEnterTransition*(this);  
 }  
  
public void supportStartPostponedEnterTransition() {  
 ActivityCompat.*startPostponedEnterTransition*(this);  
 }  
  
@Override  
 protected void onCreate(@Nullable Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_CREATE*);  
 mFragments.dispatchCreate();  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@Nullable View parent, @NonNull String name, @NonNull Context context,  
 @NonNull AttributeSet attrs) {  
 final View v = dispatchFragmentsOnCreateView(parent, name, context, attrs);  
 if (v == null) {  
 return super.onCreateView(parent, name, context, attrs);  
 }  
 return v;  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull String name, @NonNull Context context,  
 @NonNull AttributeSet attrs) {  
 final View v = dispatchFragmentsOnCreateView(null, name, context, attrs);  
 if (v == null) {  
 return super.onCreateView(name, context, attrs);  
 }  
 return v;  
 }  
  
 @Nullable  
 final View dispatchFragmentsOnCreateView(@Nullable View parent, @NonNull String name,  
 @NonNull Context context, @NonNull AttributeSet attrs) {  
 return mFragments.onCreateView(parent, name, context, attrs);  
 }  
  
@Override  
 protected void onDestroy() {  
 super.onDestroy();  
 mFragments.dispatchDestroy();  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_DESTROY*);  
 }  
  
 @Override  
 public boolean onMenuItemSelected(int featureId, @NonNull MenuItem item) {  
 if (super.onMenuItemSelected(featureId, item)) {  
 return true;  
 }  
  
 if (featureId == Window.*FEATURE\_CONTEXT\_MENU*) {  
 return mFragments.dispatchContextItemSelected(item);  
 }  
  
 return false;  
 }  
  
@Override  
 protected void onPause() {  
 super.onPause();  
 mResumed = false;  
 mFragments.dispatchPause();  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_PAUSE*);  
 }  
  
@SuppressWarnings("deprecation")  
 @Override  
 public void onStateNotSaved() {  
 mFragments.noteStateNotSaved();  
 }  
  
@Override  
 protected void onResume() {  
 mFragments.noteStateNotSaved();  
 super.onResume();  
 mResumed = true;  
 mFragments.execPendingActions();  
 }  
  
@Override  
 protected void onPostResume() {  
 super.onPostResume();  
 onResumeFragments();  
 }  
  
protected void onResumeFragments() {  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_RESUME*);  
 mFragments.dispatchResume();  
 }  
  
@Override  
 protected void onStart() {  
 mFragments.noteStateNotSaved();  
 super.onStart();  
  
 mStopped = false;  
  
 if (!mCreated) {  
 mCreated = true;  
 mFragments.dispatchActivityCreated();  
 }  
  
 mFragments.execPendingActions();  
  
 // NOTE: HC onStart goes here.  
  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_START*);  
 mFragments.dispatchStart();  
 }  
  
@Override  
 protected void onStop() {  
 super.onStop();  
  
 mStopped = true;  
 markFragmentsCreated();  
  
 mFragments.dispatchStop();  
 mFragmentLifecycleRegistry.handleLifecycleEvent(Lifecycle.Event.*ON\_STOP*);  
 }  
  
@SuppressWarnings("DeprecatedIsStillUsed")  
 @Deprecated  
 public void supportInvalidateOptionsMenu() {  
 invalidateOptionsMenu();  
 }  
  
@SuppressWarnings("deprecation")  
 @Override  
 public void dump(@NonNull String prefix, @Nullable FileDescriptor fd,  
 @NonNull PrintWriter writer, @Nullable String[] args) {  
 super.dump(prefix, fd, writer, args);  
  
 if (!shouldDumpInternalState(args)) {  
 return;  
 }  
  
 writer.print(prefix); writer.print("Local FragmentActivity ");  
 writer.print(Integer.*toHexString*(System.*identityHashCode*(this)));  
 writer.println(" State:");  
 String innerPrefix = prefix + " ";  
 writer.print(innerPrefix); writer.print("mCreated=");  
 writer.print(mCreated); writer.print(" mResumed=");  
 writer.print(mResumed); writer.print(" mStopped=");  
 writer.print(mStopped);  
  
 if (getApplication() != null) {  
 LoaderManager.*getInstance*(this).dump(innerPrefix, fd, writer, args);  
 }  
 mFragments.getSupportFragmentManager().dump(prefix, fd, writer, args);  
 }  
  
@SuppressWarnings({"unused", "DeprecatedIsStillUsed"})  
 @Deprecated  
 @MainThread  
 public void onAttachFragment(@NonNull Fragment fragment) {  
 }  
  
@NonNull  
 public FragmentManager getSupportFragmentManager() {  
 return mFragments.getSupportFragmentManager();  
 }  
  
@Deprecated  
 @NonNull  
 public LoaderManager getSupportLoaderManager() {  
 return LoaderManager.*getInstance*(this);  
 }  
  
@Override  
 @Deprecated  
 public final void validateRequestPermissionsRequestCode(int requestCode) { }  
  
 @SuppressWarnings("deprecation")  
 @CallSuper  
 @Override  
 public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,  
 @NonNull int[] grantResults) {  
 mFragments.noteStateNotSaved();  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults);  
 }  
  
public void startActivityFromFragment(@NonNull Fragment fragment,  
 @SuppressLint("UnknownNullness") Intent intent, int requestCode) {  
 startActivityFromFragment(fragment, intent, requestCode, null);  
 }  
  
@SuppressWarnings("deprecation")  
 public void startActivityFromFragment(@NonNull Fragment fragment,  
 @SuppressLint("UnknownNullness") Intent intent, int requestCode,  
 @Nullable Bundle options) {  
 // request code will be -1 if called from fragment.startActivity  
 if (requestCode == -1) {  
 ActivityCompat.*startActivityForResult*(this, intent, -1, options);  
 return;  
 }  
 // If for some reason this method is being called directly with a requestCode that is not  
 // -1, redirect it to the fragment.startActivityForResult method  
 fragment.startActivityForResult(intent, requestCode, options);  
 }  
  
@SuppressWarnings({"deprecation"})  
 @Deprecated  
 public void startIntentSenderFromFragment(@NonNull Fragment fragment,  
 @SuppressLint("UnknownNullness") IntentSender intent, int requestCode,  
 @Nullable Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags,  
 @Nullable Bundle options) throws IntentSender.SendIntentException {  
 if (requestCode == -1) {  
 ActivityCompat.*startIntentSenderForResult*(this, intent, requestCode, fillInIntent,  
 flagsMask, flagsValues, extraFlags, options);  
 return;  
 }  
 fragment.startIntentSenderForResult(intent, requestCode, fillInIntent, flagsMask,  
 flagsValues, extraFlags, options);  
 }  
  
 class HostCallbacks extends FragmentHostCallback<FragmentActivity> implements  
 OnConfigurationChangedProvider,  
 OnTrimMemoryProvider,  
 OnMultiWindowModeChangedProvider,  
 OnPictureInPictureModeChangedProvider,  
 ViewModelStoreOwner,  
 OnBackPressedDispatcherOwner,  
 ActivityResultRegistryOwner,  
 SavedStateRegistryOwner,  
 FragmentOnAttachListener,  
 MenuHost {  
  
 public HostCallbacks() {  
 super(FragmentActivity.this /\*fragmentActivity\*/);  
 }  
  
 @NonNull  
 @Override  
 public Lifecycle getLifecycle() {  
 // Instead of directly using the Activity's Lifecycle, we  
 // use a LifecycleRegistry that is nested exactly outside of  
 // when Fragments get their lifecycle changed  
 // TODO(b/127528777) Drive Fragment Lifecycle with LifecycleObserver  
 return mFragmentLifecycleRegistry;  
 }  
  
 @NonNull  
 @Override  
 public ViewModelStore getViewModelStore() {  
 return FragmentActivity.this.getViewModelStore();  
 }  
  
 @NonNull  
 @Override  
 public OnBackPressedDispatcher getOnBackPressedDispatcher() {  
 return FragmentActivity.this.getOnBackPressedDispatcher();  
 }  
  
 @Override  
 public void onDump(@NonNull String prefix, @Nullable FileDescriptor fd,  
 @NonNull PrintWriter writer, @Nullable String[] args) {  
 FragmentActivity.this.dump(prefix, fd, writer, args);  
 }  
  
 @Override  
 public boolean onShouldSaveFragmentState(@NonNull Fragment fragment) {  
 return !isFinishing();  
 }  
  
 @Override  
 @NonNull  
 public LayoutInflater onGetLayoutInflater() {  
 return FragmentActivity.this.getLayoutInflater().cloneInContext(FragmentActivity.this);  
 }  
  
 @Override  
 public FragmentActivity onGetHost() {  
 return FragmentActivity.this;  
 }  
  
 @Override  
 public void onSupportInvalidateOptionsMenu() {  
 invalidateMenu();  
 }  
  
 @Override  
 public boolean onShouldShowRequestPermissionRationale(@NonNull String permission) {  
 return ActivityCompat.*shouldShowRequestPermissionRationale*(  
 FragmentActivity.this, permission);  
 }  
  
 @Override  
 public boolean onHasWindowAnimations() {  
 return getWindow() != null;  
 }  
  
 @Override  
 public int onGetWindowAnimations() {  
 final Window w = getWindow();  
 return (w == null) ? 0 : w.getAttributes().windowAnimations;  
 }  
  
 @SuppressWarnings("deprecation")  
 @Override  
 public void onAttachFragment(@NonNull FragmentManager fragmentManager,  
 @NonNull Fragment fragment) {  
 FragmentActivity.this.onAttachFragment(fragment);  
 }  
  
 @Nullable  
 @Override  
 public View onFindViewById(int id) {  
 return FragmentActivity.this.findViewById(id);  
 }  
  
 @Override  
 public boolean onHasView() {  
 final Window w = getWindow();  
 return (w != null && w.peekDecorView() != null);  
 }  
  
 @NonNull  
 @Override  
 public ActivityResultRegistry getActivityResultRegistry() {  
 return FragmentActivity.this.getActivityResultRegistry();  
 }  
  
 @NonNull  
 @Override  
 public SavedStateRegistry getSavedStateRegistry() {  
 return FragmentActivity.this.getSavedStateRegistry();  
 }  
  
 @Override  
 public void addOnConfigurationChangedListener(  
 @NonNull Consumer<Configuration> listener  
 ) {  
 FragmentActivity.this.addOnConfigurationChangedListener(listener);  
 }  
  
 @Override  
 public void removeOnConfigurationChangedListener(  
 @NonNull Consumer<Configuration> listener  
 ) {  
 FragmentActivity.this.removeOnConfigurationChangedListener(listener);  
 }  
  
 @Override  
 public void addOnTrimMemoryListener(@NonNull Consumer<Integer> listener) {  
 FragmentActivity.this.addOnTrimMemoryListener(listener);  
 }  
  
 @Override  
 public void removeOnTrimMemoryListener(@NonNull Consumer<Integer> listener) {  
 FragmentActivity.this.removeOnTrimMemoryListener(listener);  
 }  
  
 @Override  
 public void addOnMultiWindowModeChangedListener(  
 @NonNull Consumer<MultiWindowModeChangedInfo> listener) {  
 FragmentActivity.this.addOnMultiWindowModeChangedListener(listener);  
 }  
  
 @Override  
 public void removeOnMultiWindowModeChangedListener(  
 @NonNull Consumer<MultiWindowModeChangedInfo> listener) {  
 FragmentActivity.this.removeOnMultiWindowModeChangedListener(listener);  
 }  
  
 @Override  
 public void addOnPictureInPictureModeChangedListener(  
 @NonNull Consumer<PictureInPictureModeChangedInfo> listener) {  
 FragmentActivity.this.addOnPictureInPictureModeChangedListener(listener);  
 }  
  
 @Override  
 public void removeOnPictureInPictureModeChangedListener(  
 @NonNull Consumer<PictureInPictureModeChangedInfo> listener) {  
 FragmentActivity.this.removeOnPictureInPictureModeChangedListener(listener);  
 }  
  
 @Override  
 public void addMenuProvider(@NonNull MenuProvider provider) {  
 FragmentActivity.this.addMenuProvider(provider);  
 }  
  
 @Override  
 public void addMenuProvider(@NonNull MenuProvider provider, @NonNull LifecycleOwner owner) {  
 FragmentActivity.this.addMenuProvider(provider, owner);  
 }  
  
 @Override  
 public void addMenuProvider(@NonNull MenuProvider provider, @NonNull LifecycleOwner owner,  
 @NonNull Lifecycle.State state) {  
 FragmentActivity.this.addMenuProvider(provider, owner, state);  
 }  
  
 @Override  
 public void removeMenuProvider(@NonNull MenuProvider provider) {  
 FragmentActivity.this.removeMenuProvider(provider);  
 }  
  
 @Override  
 public void invalidateMenu() {  
 FragmentActivity.this.invalidateOptionsMenu();  
 }  
 }  
  
 void markFragmentsCreated() {  
 boolean reiterate;  
 do {  
 reiterate = *markState*(getSupportFragmentManager(), Lifecycle.State.*CREATED*);  
 } while (reiterate);  
 }  
  
 private static boolean markState(FragmentManager manager, Lifecycle.State state) {  
 boolean hadNotMarked = false;  
 Collection<Fragment> fragments = manager.getFragments();  
 for (Fragment fragment : fragments) {  
 if (fragment == null) {  
 continue;  
 }  
 if (fragment.getHost() != null) {  
 FragmentManager childFragmentManager = fragment.getChildFragmentManager();  
 hadNotMarked |= *markState*(childFragmentManager, state);  
 }  
 if (fragment.mViewLifecycleOwner != null && fragment.mViewLifecycleOwner  
 .getLifecycle().getCurrentState().isAtLeast(Lifecycle.State.*STARTED*)) {  
 fragment.mViewLifecycleOwner.setCurrentState(state);  
 hadNotMarked = true;  
 }  
 if (fragment.mLifecycleRegistry.getCurrentState().isAtLeast(Lifecycle.State.*STARTED*)) {  
 fragment.mLifecycleRegistry.setCurrentState(state);  
 hadNotMarked = true;  
 }  
 }  
 return hadNotMarked;  
 }  
}

