□□□□□□□□ में कस्टम ड्राइंग का गहन विश्लेषण

यह ब्लॉग पोस्ट □□□□□□□□-4□ की सहायता से लिखा गया है।

परिचय

इस ब्लॉग पोस्ट में, हम `DrawActivity` क्लास पर चर्चा करेंगे, जो □□□□□□□□ ऐप्लिकेशन में कस्टम ड्राइंग व्यू को लागू करने का एक व्यापक उदाहरण है। हम प्रत्येक घटक और उपयोग किए गए एल्गोरिदम को विस्तार से समझाएंगे, और यह भी बताएंगे कि वे कैसे मिलकर आवश्यक कार्यक्षमता को प्राप्त करते हैं।

सामग्री सूची

□□□□□□□□□□□□□ **का अवलोकन**

`DrawActivity` वह मुख्य गतिविधि है जो ड्राइंग ऑपरेशन, इमेज क्रॉपिंग, और अन्य घटकों (जैसे □□□□□□□□□ और इमेज अपलोड) के साथ इंटरैक्शन को संभालती है। यह एक यूजर इंटरफेस प्रदान करती है जहां यूजर ड्राइंग कर सकता है, पिछले कदमों को वापस ले सकता है, फिर से कर सकता है, और इमेज को मैनिपुलेट कर सकता है।

```
public class DrawActivity extends Activity implements View.OnClickListener {
  //             ID
  public static final int CAMERA_RESULT = 1;
```

```
    public static final int CROP_RESULT = 2;
    public static final int DRAW_FRAGMENT = 0;
    public static final int RECOG_FRAGMENT = 1;
    public static final int RESULT_FRAGMENT = 2;
    public static final int WAIT_FRAGMENT = 3;
    public static final int MATERIAL_RESULT = 4;
    public static final String RESULT_JSON = "resultJson";
    public static final int INIT_FLOWER_ID = R.drawable.flower_b;
    public static final int LOGOUT = 0;
    public static final int IMAGE_RESULT = 0;

    //
    String baseUrl;
    DrawView drawView;
    Bitmap originImg;
    public static DrawActivity instance;
    View dir, clear, cameraView, materialView, scale;
    ImageView undoView, redoView;
    View upload;
    String cropPath;
    Tooltip toolTip;
    int curFragmentId = -1;
    int serverId = -1;
    private Bitmap resultBitmap;
    private RadioGroup radioGroup;
    Fragment curFragment;
    int curDrawMode;
    RadioButton drawBackBtn;
    private Activity cxt;
    Uri curPicUri;
}
```

□□□□□□□□□ **को इनिशियलाइज़ करना**

□□□□□□□□□ बनाए जाने पर, विभिन्न प्रारंभिक कार्यों को निष्पादित किया जाता है, जैसे दृश्य (□□□□) सेट करना, प्रारंभिक छवियों को लोड करना और इवेंट लिसनर्स को कॉन्फ़िगर करना।

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  instance = this;
  cxt = this;
  cropPath = PathUtils.getCropPath();
  setContentView(R.layout.draw_layout);
  findView();
  setSize();
  initOriginImage();
  toolTip = new Tooltip(this);
  initUndoRedoEnable();
  setIp();
  initDrawmode();
}
```

हिंदी व्याख्या:

यह कोड □□□□□□□□ एप्लिकेशन में `onCreate` मेथड को दर्शाता है, जो एक एक्टिविटी के बनने पर कॉल होता है। इसमें विभिन्न प्रारंभिक सेटअप कार्य किए जाते हैं, जैसे कि लेआउट सेट करना, व्यू ढूंढना, इमेज को प्रारंभ करना, और अन्य टूल्स को इनिशियलाइज़ करना। यह कोड □□□□ में लिखा गया है और इसमें □□□□□□□□ □□□ के विभिन्न क्लासेस और मेथड्स का उपयोग किया गया है।

□□□□□□□□□**()**
यह मेथड □□□□□□□□□ में उपयोग किए जाने वाले व्यूज़ को इनिशियलाइज़ करती है।

```
private void findView() {
  drawView = findViewById(R.id.drawView);
  undoView = findViewById(R.id.undo);
  redoView = findViewById(R.id.redo);
  scale = findViewById(R.id.scale);
  upload = findViewById(R.id.upload);
  clear = findViewById(R.id.clear);
  dir = findViewById(R.id.dir);
```

```
  materialView = findViewById(R.id.material);
  cameraView = findViewById(R.id.camera);

dir.setOnClickListener(this);
materialView.setOnClickListener(this);
undoView.setOnClickListener(this);
scale.setOnClickListener(this);
redoView.setOnClickListener(this);
clear.setOnClickListener(this);
cameraView.setOnClickListener(this);
upload.setOnClickListener(this);
initRadio();
}
```

यह कोड स्निपेट एक □□□□□□□□ एप्लिकेशन में □□ कंपोनेंट्स पर क्लिक लिसनर्स को सेट करता है। प्रत्येक `setOnClickListener(this)` कॉल यह इंगित करता है कि करंट क्लास (`this`) `OnClickListener` इंटरफेस को इम्प्लीमेंट करता है और उसी क्लास में `onClick` मेथड को ओवरराइड किया गया है। इसके अलावा, `initRadio()` मेथड को कॉल किया जाता है, जो संभवतः रेडियो बटन्स या अन्य □□ कंपोनेंट्स को इनिशियलाइज़ करता है।

□□□□□□□□()

ड्रॉइंग व्यू का आकार सेट करता है।

```
private void setSize() {
  setSizeByResourceSize();
  setViewSize(drawView);
}
```

(यह कोड ब्लॉक है, इसलिए इसे अनुवादित नहीं किया गया है।)

```
private void setSizeByResourceSize() {
  int width = getResources().getDimensionPixelSize(R.dimen.draw_width);
  int height = getResources().getDimensionPixelSize(R.dimen.draw_height);
  App.drawWidth = width;
  App.drawHeight = height;
}
```

इस कोड को हिंदी में समझाया जाए तो:

```
private void setSizeByResourceSize() {
  //
  int width = getResources().getDimensionPixelSize(R.dimen.draw_width);

  //
  int height = getResources().getDimensionPixelSize(R.dimen.draw_height);

  // App      drawWidth    drawHeight
  App.drawWidth = width;
  App.drawHeight = height;
}
```

यह मेथड `R.dimen.draw_width` और `R.dimen.draw_height` से चौड़ाई और ऊंचाई के आकार को प्राप्त करती है और इसे `App` वर्ग के `drawWidth` और `drawHeight` वेरिएबल्स में सेट करती है।

```
private void setViewSize(View v) {
  ViewGroup.LayoutParams lp = v.getLayoutParams();
  lp.width = App.drawWidth;
  lp.height = App.drawHeight;
  v.setLayoutParams(lp);
}
```

इस कोड को हिंदी में समझाया जाए तो:

यह एक `private` मेथड है जिसका नाम `setViewSize` है और यह एक `View` ऑब्जेक्ट को पैरामीटर के रूप में लेता है। इस मेथड का उद्देश्य `View` का आकार सेट करना है।

1. `ViewGroup.LayoutParams lp = v.getLayoutParams();` - यह लाइन `View` के लेआउट पैरामीटर्स को प्राप्त करती है और उन्हें `lp` नामक वेरिएबल में स्टोर करती है।
2. `lp.width = App.drawWidth;` - यह लाइन `View` की चौड़ाई को `App.drawWidth` के मान के बराबर सेट करती है।
3. `lp.height = App.drawHeight;` - यह लाइन `View` की ऊंचाई को `App.drawHeight` के मान के बराबर सेट करती है।
4. `v.setLayoutParams(lp);` - अंत में, यह लाइन `View` के लेआउट पैरामीटर्स को अपडेट करती है, जिससे `View` का आकार बदल जाता है।

इस प्रकार, यह मेथड `View` की चौड़ाई और ऊंचाई को `App.drawWidth` और `App.drawHeight` के मानों के अनुसार सेट कर देता है।

□□□□□□□□□□□□□□□□□**()**

ड्राइंग के लिए उपयोग की जाने वाली प्रारंभिक छवि को लोड करता है।

```
private void initOriginImage() {
  Bitmap bitmap = BitmapFactory.decodeResource(getResources(), INIT_FLOWER_ID);
  String imgPath = PathUtils.getCameraPath();
  BitmapUtils.saveBitmapToPath(bitmap, imgPath);
  Uri uri1 = Uri.fromFile(new File(imgPath));
  setImageByUri(uri1);
}
```

यह कोड एक मूल छवि को प्रारंभ करने के लिए है। इसमें `BitmapFactory.decodeResource` का उपयोग करके एक बिटमैप बनाया जाता है, फिर `PathUtils.getCameraPath` का उपयोग करके छवि का पथ प्राप्त किया जाता है। इसके बाद `BitmapUtils.saveBitmapToPath` का उपयोग करके बिटमैप को उस पथ पर सहेजा जाता है। अंत में, `Uri.fromFile` का उपयोग करके फ़ाइल से □□□ बनाया जाता है और `setImageByUri` मेथड का उपयोग करके छवि सेट की जाती है।

छवि संचालन को संभालना

□□□□□□□□ विभिन्न छवि संचालनों को संभालता है, जैसे □□□ के माध्यम से छवि सेट करना, क्रॉप करना और खींची गई बिटमैप को सहेजना।

□□□□□□□□□□□□□□□**(**□□□ □□□**)**

दिए गए □□□ से छवि को लोड करें और ड्राइंग के लिए तैयार करें।

```
private void setImageByUri(final Uri uri) {
  new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
      curPicUri = uri;
      Bitmap bitmap = null;
      try {
        if (uri != null) {
          bitmap = BitmapUtils.getBitmapByUri(DrawActivity.this, uri);
        }
      } catch (Exception e) {
```

```
            e.printStackTrace();
        }
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग कोड है और इसे अनुवादित करने की आवश्यकता नहीं है।)

```
int originW = bitmap.getWidth();
int originH = bitmap.getHeight();
if (originW != App.drawWidth || originH != App.drawHeight) {
    float originRadio = originW * 1.0f / originH;
    float radio = App.drawWidth * 1.0f / App.drawHeight;
    if (Math.abs(originRadio - radio) < 0.01) {
        Bitmap originBm = bitmap;
        bitmap = Bitmap.createScaledBitmap(originBm, App.drawWidth, App.drawHeight, false);
        originBm.recycle();
    } else {
        cropIt(uri);
        return;
    }
}
ImageLoader imageLoader = ImageLoader.getInstance();
imageLoader.addOrReplaceToMemoryCache("origin", bitmap);
originImg = bitmap;
serverId = -1;
```

हिंदी अनुवाद:

```
int originW = bitmap.getWidth();
int originH = bitmap.getHeight();
if (originW != App.drawWidth || originH != App.drawHeight) {
    float originRadio = originW * 1.0f / originH;
    float radio = App.drawWidth * 1.0f / App.drawHeight;
    if (Math.abs(originRadio - radio) < 0.01) {
        Bitmap originBm = bitmap;
        bitmap = Bitmap.createScaledBitmap(originBm, App.drawWidth, App.drawHeight, false);
        originBm.recycle();
    } else {
        cropIt(uri);
```

```
        return;
    }
}
ImageLoader imageLoader = ImageLoader.getInstance();
imageLoader.addOrReplaceToMemoryCache("origin", bitmap);
originImg = bitmap;
serverId = -1;
```

व्याख्या:

- originW और originH बिटमैप की मूल चौड़ाई और ऊंचाई को दर्शाते हैं।
- यदि बिटमैप की चौड़ाई और ऊंचाई App.drawWidth और App.drawHeight के बराबर नहीं है, तो अनुपात की जांच की जाती है।
- यदि अनुपात लगभग समान है, तो बिटमैप को नए आकार में स्केल किया जाता है और पुराने बिटमैप को रीसायकल किया जाता है।
- यदि अनुपात समान नहीं है, तो cropIt(uri) फ़ंक्शन को कॉल किया जाता है और फ़ंक्शन से वापस लौटा जाता है।
- अंत में, बिटमैप को ImageLoader के मेमोरी कैश में जोड़ा जाता है और originImg और serverId को सेट किया जाता है।

```
drawView.setSrcBitmap(originImg);
showDrawFragment(App.ALL_INFO);
curDrawMode = App.DRAW_FORE;
}
}, 500);
}
```

( )
छवि क्रॉपिंग गतिविधि शुरू करता है।

```
public void cropIt(Uri uri) {
  Crop.startPhotoCrop(this, uri, cropPath, CROP_RESULT);
}
```

(: .)

()
ड्रॉ किए गए बिटमैप को फ़ाइल में सहेजें और सर्वर पर अपलोड करें।

```java
public void saveBitmap() {
  Bitmap handBitmap = drawView.getHandBitmap();
  Bitmap originBitmap = drawView.getSrcBitmap();
  saveBitmapToFileAndUpload(handBitmap, originBitmap);
}
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग भाषा का हिस्सा है और इसे अपरिवर्तित रहना चाहिए।)

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□(□□□□□□□ □□□□□□□□□□□, □□□□□□□ □□□□□□□□□□□□□□)
बिटमैप को फ़ाइल में सहेजें और एसिंक्रोनस रूप से अपलोड करें।

```java
private void saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap) {
  final String originPath = PathUtils.getOriginPath();
  BitmapUtils.saveBitmapToPath(originBitmap, originPath);
  final String handPath = PathUtils.getHandPath();
  BitmapUtils.saveBitmapToPath(handBitmap, handPath);
  new AsyncTask<Void, Void, Void>() {
    boolean res;
    Bitmap foreBitmap;
    Bitmap backBitmap;
```

(नोट: कोड ब्लॉक को हिंदी में ट्रांसलेट नहीं किया जाता है क्योंकि यह प्रोग्रामिंग सिंटैक्स है और इसे अपरिवर्तित रखना आवश्यक है।)

```java
@Override
protected void onPreExecute() {
  super.onPreExecute();
  showWaitFragment();
}

@Override
protected Void doInBackground(Void... params) {
  try {
    if (baseUrl == null) {
      throw new Exception("baseUrl is null");
    }
    String jsonRes = UploadImage.upload(baseUrl, serverId, Web.STATUS_CONTINUE, originPath, handPath, nu
    getJsonData(jsonRes);
    res = true;
```

```
  } catch (Exception e) {
    res = false;
    e.printStackTrace();
  }
  return null;
}

private void getJsonData(String jsonRes) throws Exception {
  JSONObject json = new JSONObject(jsonRes);
  if (serverId == -1) {
    serverId = json.getInt(Web.ID);
  }
  String foreUrl = json.getString(Web.FORE);
  String backUrl = json.getString(Web.BACK);
  String resultUrl = json.getString(Web.RESULT);
  foreBitmap = Web.getBitmapFromUrlByStream1(foreUrl, 0);
  backBitmap = Web.getBitmapFromUrlByStream1(backUrl, 0);
  resultBitmap = Web.getBitmapFromUrlByStream1(resultUrl, 0);
}

@Override
protected void onPostExecute(Void aVoid) {
  super.onPostExecute(aVoid);
  if (res) {
    showRecogFragment(foreBitmap, backBitmap);
  } else {
    Utils.toast(DrawActivity.this, R.string.server_error);
    recogNo();
  }
}

}.□□□□□□□□(); }

---

###
```

Activity , - fragment

showDrawFragment(int infoId)

```java
private void showDrawFragment(int infoId) {
  curFragmentId = DRAW_FRAGMENT;
  curFragment = new DrawFragment(infoId);
  showFragment(curFragment);
}
```

यह कोड एक □□□□ मेथड `showDrawFragment` को दिखाता है जो एक `DrawFragment` को प्रदर्शित करता है। इस मेथड में `infoId` नामक एक पैरामीटर लिया जाता है।

1. `curFragmentId` को `DRAW_FRAGMENT` के रूप में सेट किया जाता है।
2. `curFragment` को `DrawFragment` का एक नया उदाहरण बनाकर `infoId` पास करके इनिशियलाइज़ किया जाता है।
3. अंत में, `showFragment` मेथड को कॉल करके `curFragment` को प्रदर्शित किया जाता है।

इस कोड को हिंदी में समझाने के लिए, यह एक फ़ंक्शन है जो एक विशेष प्रकार के फ़्रैगमेंट (`DrawFragment`) को दिखाने के लिए उपयोग किया जाता है। यह फ़ंक्शन एक आईडी (`infoId`) लेता है और उस आईडी के आधार पर एक नया फ़्रैगमेंट बनाता है और उसे प्रदर्शित करता है।

□□□□□□□□□□□□□□□□□□**()**
प्रतीक्षा फ़्रैगमेंट दिखाएं।

```java
private void showWaitFragment() {
  curFragmentId = WAIT_FRAGMENT;
  showFragment(new WaitFragment());
}
```

(नोट: कोड ब्लॉक्स को अनुवादित नहीं किया जाता है, क्योंकि वे प्रोग्रामिंग भाषा में लिखे गए होते हैं और उनका अनुवाद करने की आवश्यकता नहीं होती है।)

□□□□□□□□□□□□□**(□□□□□□□□□ □□□□□□□□□)**
निर्दिष्ट फ़्रैगमेंट के साथ वर्तमान फ़्रैगमेंट को बदलें।

```java
private void showFragment(Fragment fragment) {
  FragmentTransaction trans = getFragmentManager().beginTransaction();
```

```
  trans.replace(R.id.rightLayout, fragment);
  trans.commit();
}
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग भाषा का हिस्सा है और इसे बदलने की आवश्यकता नहीं है।)

इवेंट हैंडलिंग

□□□□□□□□□ उपयोगकर्ता के विभिन्न इंटरैक्शन को संभालता है, जैसे बटन क्लिक और मेनू चयन।

□□□□□□□□**(**□□□□ □**)**
विभिन्न दृश्यों (व्यू) पर क्लिक इवेंट को संभालता है।

```
@Override
public void onClick(View v) {
  int id = v.getId();
  if (id == R.id.drawOk) {
    if (drawView.isDrawFinish()) {
      saveBitmap();
    } else {
      Utils.alertDialog(this, R.string.please_draw_finish);
    }
  } else if (id == R.id.recogOk) {
    recogOk();
  } else if (id == R.id.recogNo) {
    recogNo();
  } else if (id == R.id.dir) {
    Utils.getGalleryPhoto(this, IMAGE_RESULT);
  } else if (id == R.id.clear) {
    clearEverything();
  } else if (id == R.id.undo) {
    drawView.undo();
  } else if (id == R.id.redo) {
    drawView.redo();
  } else if (id == R.id.camera) {
```

```
    Utils.takePhoto(cxt, CAMERA_RESULT);
  } else if (id == R.id.material) {
    goMaterial();
  } else if (id == R.id.upload) {
    com.lzw.commons.Utils.goActivity(cxt, PhotoActivity.class);
  } else if (id == R.id.scale) {
    cropIt(curPicUri);
  }
}
```

हिंदी व्याख्या:

यह कोड एक `onClick` मेथड है जो `View` के क्लिक इवेंट को हैंडल करता है। यहां `View` के आईडी के आधार पर अलग-अलग एक्शन्स को परफॉर्म किया जाता है।

- यदि `id` `R.id.drawOk` है और `drawView` में ड्रॉइंग पूरी हो चुकी है, तो `saveBitmap()` मेथड को कॉल किया जाता है। अगर ड्रॉइंग पूरी नहीं हुई है, तो एक अलर्ट डायलॉग दिखाया जाता है।
- यदि `id` `R.id.recogOk` है, तो `recogOk()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.recogNo` है, तो `recogNo()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.dir` है, तो गैलरी से फोटो चुनने के लिए `Utils.getGalleryPhoto()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.clear` है, तो `clearEverything()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.undo` है, तो `drawView.undo()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.redo` है, तो `drawView.redo()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.camera` है, तो कैमरा से फोटो लेने के लिए `Utils.takePhoto()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.material` है, तो `goMaterial()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.upload` है, तो `PhotoActivity` को शुरू करने के लिए `com.lzw.commons.Utils.goActivity()` मेथड को कॉल किया जाता है।
- यदि `id` `R.id.scale` है, तो `cropIt(curPicUri)` मेथड को कॉल किया जाता है।

(, ,)
दूसरी गतिविधियों के परिणामों को संभालता है, जैसे कि छवि चयन या क्रॉपिंग।

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (resultCode != RESULT_CANCELED) {
    Uri uri;
    switch (requestCode) {
      case IMAGE_RESULT:
```

```
        if (data != null) {
          setImageByUri(data.getData());
        }
        break;
      case CAMERA_RESULT:
        setImageByUri(Utils.getCameraUri());
        break;
      case CROP_RESULT:
        uri = Uri.fromFile(new File(cropPath));
        setImageByUri(uri);
        break;
      case MATERIAL_RESULT:
        setImageByUri(data.getData());
    }
  }
}
```

पूर्ववत और पुनः करें सुविधा

□□□□□□□□ ड्राइंग ऑपरेशन के लिए पूर्ववत (□□□□) और पुनः करें (□□□□) सुविधाएं प्रदान करता है।

□□□□□□□□□□□□□□□□□□□□**()**
कॉलबैक फ़ंक्शन सेट करके पूर्ववत (□□□□) और पुनः करें (□□□□) सुविधाओं को प्रारंभ करता है।

```
void initUndoRedoEnable() {
  drawView.history.setCallBack(new History.CallBack() {
    @Override
    public void onHistoryChanged() {
      setUndoRedoEnable();
      if (curFragmentId != DRAW_FRAGMENT) {
        showDrawFragment(curDrawMode);
      }
    }
  });
}
```

यह कोड `initUndoRedoEnable` नामक एक मेथड को परिभाषित करता है जो `drawView.history` के लिए एक कॉलबैक सेट करता है। जब भी इतिहास (□□□□□□□□) में कोई परिवर्तन होता है, तो `onHistoryChanged` मेथड कॉल होती है। इस मेथड में, `setUndoRedoEnable` फ़ंक्शन को कॉल किया जाता है और यह जाँचा जाता है कि वर्तमान फ़्रैगमेंट (□□□□□□□□□□□□□□□) ड्रॉ फ़्रैगमेंट (□□□□_□□□□□□□□□) नहीं है। यदि नहीं है, तो `showDrawFragment` फ़ंक्शन को वर्तमान ड्रॉ मोड (□□□□□□□□□□□□□) के साथ कॉल किया जाता है।

```
void setUndoRedoEnable() {
  redoView.setEnabled(drawView.history.canRedo());
  undoView.setEnabled(drawView.history.canUndo());
}
```

कस्टम □□□□□□□□□

`DrawView` एक कस्टम व्यू है जो ड्राइंग ऑपरेशन, टच इवेंट्स और ज़ूमिंग को संभालने के लिए उपयोग किया जाता है।

□□□□□□□□□□□□□□**(**□□□□□□□□□□□□□ □□□□□**)**
ड्रॉइंग और ज़ूम करने के लिए टच इवेंट्स को संभालता है।

```
@Override
public boolean onTouchEvent(MotionEvent event) {
  if (!scaleMode) {
    handleDrawTouchEvent(event);
  } else {
    handleScaleTouchEvent(event);
  }
  return true;
}
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया जाता है क्योंकि यह प्रोग्रामिंग भाषा का हिस्सा है और इसे अपरिवर्तित रखना चाहिए।)

```
private void handleDrawTouchEvent(MotionEvent event) {
  int action = event.getAction();
  float x = event.getX();
  float y = event.getY();
  if (action == MotionEvent.ACTION_DOWN) {
    path.moveTo(x, y);
```

```
  } else if (action == MotionEvent.ACTION_MOVE) {
    path.quadTo(preX, preY, x, y);
  } else if (action == MotionEvent.ACTION_UP) {
    Matrix matrix1 = new Matrix();
    matrix.invert(matrix1);
    path.transform(matrix1);
    paint.setStrokeWidth(strokeWidth * 1.0f / totalRatio);
    history.saveToStack(path, paint);
    cacheCanvas.drawPath(path, paint);
    paint.setStrokeWidth(strokeWidth);
    path.reset();
  }
  setPrev(event);
  invalidate();
}
```

यह कोड □□□□□□□□ में टच इवेंट को हैंडल करने के लिए है, जिसमें उपयोगकर्ता स्क्रीन पर ड्रॉ कर सकता है। यहां `MotionEvent` के माध्यम से टच की स्थिति और एक्शन को पकड़ा जाता है। यदि उपयोगकर्ता स्क्रीन को छूता है (`ACTION_DOWN`), तो पथ (□□□□) उस बिंदु पर शुरू होता है। यदि उपयोगकर्ता अपनी उंगली को स्क्रीन पर घुमाता है (`ACTION_MOVE`), तो पथ को उसके पिछले और वर्तमान स्थान के बीच एक वक्र के रूप में खींचा जाता है। जब उपयोगकर्ता अपनी उंगली को स्क्रीन से उठाता है (`ACTION_UP`), तो पथ को ट्रांसफॉर्म किया जाता है, पेंट की स्ट्रोक चौड़ाई को एडजस्ट किया जाता है, और पथ को कैनवास पर ड्रॉ किया जाता है। अंत में, पथ को रीसेट कर दिया जाता है और स्क्रीन को अपडेट करने के लिए `invalidate()` कहा जाता है।

```
private void handleScaleTouchEvent(MotionEvent event) {
  switch (event.getActionMasked()) {
    case MotionEvent.ACTION_POINTER_DOWN:
      lastFingerDist = calFingerDistance(event);
      break;
    case MotionEvent.ACTION_MOVE:
      if (event.getPointerCount() == 1) {
        handleMove(event);
      } else if (event.getPointerCount() == 2) {
        handleZoom(event);
      }
      break;
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
```

```
      lastMoveX = -1;
      lastMoveY = -1;
      break;
    default:
      break;
  }
}
```

हिंदी अनुवाद:

```
private void handleScaleTouchEvent(MotionEvent event) {
  switch (event.getActionMasked()) {
    case MotionEvent.ACTION_POINTER_DOWN:
      lastFingerDist = calFingerDistance(event);
      break;
    case MotionEvent.ACTION_MOVE:
      if (event.getPointerCount() == 1) {
        handleMove(event);
      } else if (event.getPointerCount() == 2) {
        handleZoom(event);
      }
      break;
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
      lastMoveX = -1;
      lastMoveY = -1;
      break;
    default:
      break;
  }
}
```

व्याख्या:

- `MotionEvent.ACTION_POINTER_DOWN`: जब एक नया टच पॉइंट (उंगली) स्क्रीन पर दबाया जाता है, तो `lastFingerDist` को उंगलियों के बीच की दूरी के साथ अपडेट किया जाता है।
- `MotionEvent.ACTION_MOVE`: यदि स्क्रीन पर केवल एक उंगली है, तो `handleMove(event)` को कॉल किया जाता है। यदि दो उंगलियां हैं, तो `handleZoom(event)` को कॉल किया जाता है।

- MotionEvent.ACTION_UP और MotionEvent.ACTION_POINTER_UP: जब उंगली स्क्रीन से हट जाती है, तो lastMoveX और lastMoveY को रीसेट किया जाता है।
- default: अन्य सभी केसों के लिए कोई विशेष कार्रवाई नहीं की जाती है।

```
private void handleMove(MotionEvent event) {
  float moveX = event.getX();
  float moveY = event.getY();
  if (lastMoveX == -1 && lastMoveY == -1) {
    lastMoveX = moveX;
    lastMoveY = moveY;
  }
  moveDistX = (int) (moveX - lastMoveX);
  moveDistY = (int) (moveY - lastMoveY);
  if (moveDistX + totalTranslateX > 0 || moveDistX + totalTranslateX + curBitmapWidth < width) {
    moveDistX = 0;
  }
  if (moveDistY + totalTranslateY > 0 || moveDistY + totalTranslateY + curBitmapHeight < height) {
    moveDistY = 0;
  }
  status = STATUS_MOVE;
  invalidate();
  lastMoveX = moveX;
  lastMoveY = moveY;
}
```

इस कोड में, handleMove मेथड का उपयोग यूजर के टच मूवमेंट को हैंडल करने के लिए किया जाता है। यह मेथड MotionEvent से X और Y कोऑर्डिनेट्स प्राप्त करता है और उन्हें पिछले कोऑर्डिनेट्स के साथ तुलना करता है। यदि यूजर का मूवमेंट इमेज की सीमाओं से बाहर जाता है, तो मूवमेंट को रोक दिया जाता है। अंत में, invalidate() को कॉल करके व्यू को रीड्रॉ किया जाता है और पिछले कोऑर्डिनेट्स को अपडेट किया जाता है।

```
private void handleZoom(MotionEvent event) {
  float fingerDist = calFingerDistance(event);
  calFingerCenter(event);
  if (fingerDist > lastFingerDist) {
    status = STATUS_ZOOM_OUT;
  } else {
    status = STATUS_ZOOM_IN;
```

```
  }
  scaledRatio = fingerDist * 1.0f / lastFingerDist;
  totalRatio = totalRatio * scaledRatio;
  if (totalRatio < initRatio) {
    totalRatio = initRatio;
  } else if (totalRatio > initRatio * 4) {
    totalRatio = initRatio * 4;
  }
  lastFingerDist = fingerDist;
  invalidate();
}
```

हिंदी अनुवाद:

```
private void handleZoom(MotionEvent event) {
  float fingerDist = calFingerDistance(event);
  calFingerCenter(event);
  if (fingerDist > lastFingerDist) {
    status = STATUS_ZOOM_OUT;
  } else {
    status = STATUS_ZOOM_IN;
  }
  scaledRatio = fingerDist * 1.0f / lastFingerDist;
  totalRatio = totalRatio * scaledRatio;
  if (totalRatio < initRatio) {
    totalRatio = initRatio;
  } else if (totalRatio > initRatio * 4) {
    totalRatio = initRatio * 4;
  }
  lastFingerDist = fingerDist;
  invalidate();
}
```

व्याख्या:

- `handleZoom` मेथड का उपयोग ज़ूम इन और ज़ूम आउट को संभालने के लिए किया जाता है।
- `calFingerDistance` मेथड दो उंगलियों के बीच की दूरी की गणना करता है।

- `calFingerCenter` मेथड दो उंगलियों के केंद्र बिंदु की गणना करता है।
- यदि उंगलियों की दूरी पिछली दूरी से अधिक है, तो `STATUS_ZOOM_OUT` सेट किया जाता है, अन्यथा `STATUS_ZOOM_IN` सेट किया जाता है।
- `scaledRatio` और `totalRatio` का उपयोग ज़ूम स्तर को समायोजित करने के लिए किया जाता है।
- `totalRatio` को `initRatio` और `initRatio * 4` के बीच सीमित किया जाता है।
- अंत में, `invalidate()` को कॉल करके व्यू को पुनः ड्रा करने के लिए कहा जाता है।

( )

व्यू की वर्तमान स्थिति को ड्रॉ करता है।

```
@Override
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  if (scaleMode) {
    switch (status) {
      case STATUS_MOVE:
        move(canvas);
        break;
      case STATUS_ZOOM_IN:
      case STATUS_ZOOM_OUT:
        zoom(canvas);
        break;
      default:
        if (cacheBm != null) {
          canvas.drawBitmap(cacheBm, matrix, null);
          canvas.drawPath(path, paint);
        }
    }
  } else {
    if (cacheBm != null) {
      canvas.drawBitmap(cacheBm, matrix, null);
      canvas.drawPath(path, paint);
    }
  }
}
```

हिंदी व्याख्या:

यह कोड एक `onDraw` मेथड को दिखाता है जो `Canvas` पर ड्रॉइंग करने के लिए उपयोग किया जाता है। यह मेथड `scaleMode` के आधार पर अलग-अलग कार्य करता है:

1. □□□□□□□□□□ **सक्रिय होने पर:**
   - □ यदि `status` `STATUS_MOVE` है, तो `move(canvas)` मेथड को कॉल किया जाता है।
   - □ यदि `status` `STATUS_ZOOM_IN` या `STATUS_ZOOM_OUT` है, तो `zoom(canvas)` मेथड को कॉल किया जाता है।
   - □ अन्य स्थितियों में, यदि `cacheBm` (कैश्ड बिटमैप) `null` नहीं है, तो `canvas` पर बिटमैप और पाथ को ड्रॉ किया जाता है।
2. □□□□□□□□□□ **निष्क्रिय होने पर:**
   - □ यदि `cacheBm` `null` नहीं है, तो `canvas` पर बिटमैप और पाथ को ड्रॉ किया जाता है।

यह कोड मुख्य रूप से ग्राफिक्स को हेरफेर करने और डिस्प्ले करने के लिए उपयोग किया जाता है, जैसे कि इमेज को मूव करना या ज़ूम करना।

□□□□(□□□□□□□ □□□□□□□)
स्केलिंग के दौरान मूवमेंट ऑपरेशन को संभालता है।

```
private void move(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  totalTranslateX = moveDistX + totalTranslateX;
  totalTranslateY = moveDistY + totalTranslateY;
  matrix.postTranslate(totalTranslateX, totalTranslateY);
  canvas.drawBitmap(cacheBm, matrix, null);
}
```

यह कोड □□□□ में लिखा गया है और इसमें `Canvas` पर एक बिटमैप को स्थानांतरित (□□□□) करने का तरीका दिखाया गया है। इस कोड को हिंदी में समझाने की कोशिश करते हैं:

1. **□□□□□□.□□□□□□();**
   यह `matrix` को रीसेट करता है, यानी इसे अपने डिफ़ॉल्ट स्थिति में ले आता है।
2. **□□□□□□.□□□□□□□□□□(□□□□□□□□□□□, □□□□□□□□□□□);**
   यह `matrix` को `totalRatio` के अनुसार स्केल करता है। `totalRatio` एक स्केल फैक्टर है जो बिटमैप के आकार को बदलता है।
3. **□□□□□□□□□□□□□□□□ = □□□□□□□□□□ + □□□□□□□□□□□□□□□□;**
   यह `totalTranslateX` को अपडेट करता है। `moveDistX` वह दूरी है जिससे बिटमैप को □-अक्ष पर स्थानांतरित किया जाना है।

4. □□□□□□□□□□□□□□□□ = □□□□□□□□□□ + □□□□□□□□□□□□□□□□;

   यह `totalTranslateY` को अपडेट करता है। `moveDistY` वह दूरी है जिससे बिटमैप को □-अक्ष पर स्थानांतरित किया जाना है।

5. □□□□□□□.□□□□□□□□□□□□□□□□(□□□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□);

   यह `matrix` को `totalTranslateX` और `totalTranslateY` के अनुसार स्थानांतरित करता है।

6. □□□□□□□.□□□□□□□□□□□□□(□□□□□□□□□, □□□□□□□, □□□□□);

   अंत में, यह `cacheBm` (कैश्ड बिटमैप) को `canvas` पर `matrix` के अनुसार ड्रा करता है।

इस तरह, यह कोड बिटमैप को स्केल और स्थानांतरित करके `Canvas` पर प्रदर्शित करता है।

□□□□□(□□□□□□□ □□□□□□□)

ज़ूम ऑपरेशन को संभालता है।

```
private void zoom(Canvas canvas) {
  matrix.reset();
  matrix.postScale(totalRatio, totalRatio);
  int scaledWidth = (int) (cacheBm.getWidth() * totalRatio);
  int scaledHeight = (int) (cacheBm.getHeight() * totalRatio);
  int translateX;
  int translateY;
  if (curBitmapWidth < width) {
    translateX = (width - scaledWidth) / 2;
  } else {
    translateX = (int) (centerPointX + (totalTranslateX - centerPointX) * scaledRatio);
    if (translateX > 0) {
      translateX = 0;
    } else if (scaledWidth + translateX < width) {
      translateX = width - scaledWidth;
    }
  }
  if (curBitmapHeight < height) {
    translateY = (height - scaledHeight) / 2;
  } else {
    translateY = (int) (centerPointY + (totalTranslateY - centerPointY) * scaledRatio);
    if (translateY > 0) {
      translateY = 0;
```

```
    } else if (scaledHeight + translateY < height) {
      translate
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग भाषा का हिस्सा है और इसे बदलने की आवश्यकता नहीं है।)

□ = □□□□□□□ - □□□□□□□□□□□□□□□; } } □□□□□□□□□□□□□□□□□□ = □□□□□□□□□□□□; □□□□□□□□□□□□□□□□□□ = □□□□□□□□□□□□; □□□□□□□□□□□□□□□□□□ = □□□□□□□□□□□□□; □□□□□□□□□□□□□□□□□□ = □□□□□□□□□□□□□□□; □□□□□.□□□□□□□□□□□□□□□□(□□□□□□□□□□□□□, □□□□□□□□□□□□□); □□□□□□□.□□□□□□□□□□□□□(□□□□□□□□□, □□□□□□□, □□□□□); }

###

`History`   (undo)   (redo)

saveToStack(Path path, Paint paint)

```java
public void saveToStack(Path path, Paint paint) {
  Draw draw = new Draw();
  draw.path = new Path(path);
  draw.paint = new Paint(paint);
  saveToStack(draw);
}
```

यह कोड एक `saveToStack` मेथड को परिभाषित करता है जो एक `Path` और `Paint` ऑब्जेक्ट को लेता है और उन्हें `Draw` ऑब्जेक्ट में सहेजता है। इसके बाद यह `Draw` ऑब्जेक्ट को `saveToStack` मेथड में पास करता है।

```
public void saveToStack(Draw draw) {
  curPos++;
  while (histroy.size() > curPos) {
    histroy.pop();
  }
  histroy.push(draw);
  if (callBack != null) {
    callBack.onHistoryChanged();
```

```
  }
}
```

यह कोड एक `Draw` ऑब्जेक्ट को स्टैक में सेव करता है और इतिहास (□□□□□□□□) को अपडेट करता है। यदि `callBack` ऑब्जेक्ट उपलब्ध है, तो यह `onHistoryChanged` मेथड को कॉल करता है।

□□□□□□□□□□□□□□□□(□□□ □)

इतिहास में दिए गए बिंदु की स्थिति को दर्शाने वाला बिटमैप लौटाता है।

```
public Bitmap getBitmapAtDraw(int n) {
  Canvas canvas = new Canvas();
  Bitmap bm = Utils.getCopyBitmap(srcBitmap);
  canvas.setBitmap(bm);
  for (int i = 0; i <= n; i++) {
    Draw draw = histroy.get(i);
    canvas.drawPath(draw.path, draw.paint);
  }
  return bm;
}
```

(यह कोड ब्लॉक को हिंदी में ट्रांसलेट नहीं किया गया है क्योंकि यह प्रोग्रामिंग कोड है और इसे ट्रांसलेट करने की आवश्यकता नहीं है।)

□□□□()

पिछले कार्य को पूर्ववत करने के लिए क्रियान्वित करें।

```
public Bitmap undo() throws UnsupportedOperationException {
  if (canUndo()) {
    curPos--;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("                    ");
  }
}
```

□□□□()

रीडू (पुनः करना) ऑपरेशन को निष्पादित करता है।

```
public Bitmap redo() throws UnsupportedOperationException {
  if (canRedo()) {
    curPos++;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("                    ");
  }
}
```

□□□□□□□□**()**
जाँचता है कि क्या पिछली क्रिया को पूर्ववत किया जा सकता है।

```
public boolean canUndo() {
  return curPos > 0;
}
```

(यह कोड हिंदी में अनुवाद नहीं किया जा सकता क्योंकि यह एक प्रोग्रामिंग भाषा का कोड है और इसे मूल रूप में ही रखना चाहिए।)

□□□□□□□□**()**
जाँच करता है कि क्या पुनः करना संभव है।

```
public boolean canRedo() {
  return curPos + 1 < histroy.size();
}
```

निष्कर्ष

DrawActivity और इससे संबंधित कक्षाएँ □□□□□□□□ में कस्टम ड्राइंग व्यू (□□□□□□□ □□□□□□□□ □□□□) को लागू करने के लिए एक व्यापक उदाहरण प्रदान करती हैं। यह विभिन्न तकनीकों को प्रदर्शित करता है, जैसे टच इवेंट्स को संभालना, ड्राइंग इतिहास को प्रबंधित करना, और अन्य घटकों (जैसे □□□□□□□□□ और □□□□□□□□□□□□□ □□□□□□) के साथ एकीकरण। प्रत्येक घटक और एल्गोरिदम को समझकर, आप इन तकनीकों का उपयोग अपने ऐप्लिकेशन में शक्तिशाली और इंटरैक्टिव ड्राइंग सुविधाएँ बनाने के लिए कर सकते हैं।