Exploración Profunda del Dibujo Personalizado en Android

Esta entrada del blog fue escrita con la ayuda de ChatGPT-4o.

Introducción

En esta entrada de blog, exploraremos la clase `DrawActivity`, un ejemplo completo que implementa una vista de dibujo personalizada en una aplicación Android. Desglosaremos cada componente y los algoritmos utilizados, explicando en detalle cómo trabajan juntos para lograr la funcionalidad deseada.

Índice

Resumen de DrawActivity

DrawActivity es una actividad principal en la aplicación, que permite a los usuarios dibujar y editar imágenes. A continuación, se presenta una descripción general de sus características y funcionalidades clave.

Características principales

1. **Interfaz de dibujo intuitiva**: La interfaz de usuario está diseñada para ser fácil de usar, con herramientas de dibujo accesibles y una experiencia de usuario fluida.
2. **Herramientas de dibujo**: Incluye una variedad de herramientas como pinceles, lápices, formas geométricas y opciones de color.
3. **Capas**: Los usuarios pueden trabajar con múltiples capas, lo que permite una edición no destructiva y una mayor flexibilidad en el diseño.
4. **Guardado y exportación**: Las imágenes creadas pueden guardarse en el dispositivo o exportarse en varios formatos, como PNG o JPEG.
5. **Deshacer y rehacer**: Funcionalidades de deshacer y rehacer para corregir errores o revisar cambios anteriores.

Funcionalidades avanzadas

- **Zoom y desplazamiento**: Permite a los usuarios acercar y alejar la imagen, así como desplazarse por el lienzo.
- **Selección y transformación**: Herramientas para seleccionar y transformar partes de la imagen, como mover, rotar y escalar.
- **Filtros y efectos**: Aplicación de filtros y efectos para mejorar o modificar la apariencia de la imagen.

Ejemplo de código A continuación, se muestra un ejemplo básico de cómo se podría inicializar la actividad `DrawActivity` en una aplicación Android:

```
public class DrawActivity extends AppCompatActivity {
    private DrawingView drawingView;

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

        drawingView = findViewById(R.id.drawingView);
        // Configuración inicial del lienzo y herramientas
    }
}
```

Este código establece la actividad principal y configura la vista de dibujo (`DrawingView`) donde los usuarios pueden comenzar a crear sus obras de arte.

Conclusión DrawActivity es una herramienta poderosa para la creación y edición de imágenes, diseñada para ser accesible tanto para principiantes como para usuarios avanzados. Con su conjunto completo de características y una interfaz intuitiva, es una opción ideal para cualquier persona interesada en el diseño gráfico móvil.

`DrawActivity` es la actividad principal que maneja las operaciones de dibujo, el recorte de imágenes y la interacción con otros componentes, como fragmentos y la carga de imágenes. Proporciona una interfaz de usuario donde los usuarios pueden dibujar, deshacer, rehacer y manipular imágenes.

```
public class DrawActivity extends Activity implements View.OnClickListener {
  // Constantes para códigos de solicitud y IDs de fragmentos
  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;

  // Variables para manejar imágenes y operaciones de dibujo
  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;
}
```

Inicialización de Activity

Al crear la Activity, se ejecutan varias operaciones de inicialización, como configurar las vistas, cargar imágenes iniciales y configurar los escuchadores de eventos.

```
@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();
}
```

Nota: El código anterior es un bloque de código Java y no necesita ser traducido, ya que los nombres de métodos, variables y clases deben permanecer en inglés para mantener la funcionalidad del programa.

findView()

Este método inicializa las vistas utilizadas en la Activity.

```
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();
}
```

setSize()

Establece el tamaño de la vista de dibujo.

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

El código anterior define un método privado llamado `setSize()` en Java. Este método realiza dos acciones principales:

1. `setSizeByResourceSize();`: Llama a un método que establece el tamaño basado en el tamaño de algún recurso.

2. `setViewSize(drawView);`: Llama a otro método que establece el tamaño de una vista específica, en este caso, `drawView`.

En resumen, este método se encarga de configurar el tamaño de un componente o vista en una aplicación Java, utilizando primero un recurso y luego aplicándolo a una vista específica.

```
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;
}

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

initOriginImage()

Carga la imagen inicial que se utilizará para el dibujo.

```
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);
}
```

Manipulación de Imágenes

La actividad maneja varias operaciones de imágenes, como establecer imágenes a través de URI, recortar y guardar mapas de bits dibujados.

setImageByUri(Uri uri)

Carga una imagen desde el URI proporcionado y la prepara para el dibujo.

```
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();
      }

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;

drawView.setSrcBitmap(originImg);
```

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

cropIt(Uri uri)

Inicia la actividad de recorte de imágenes.

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

saveBitmap()

Guarda el mapa de bits dibujado en un archivo y lo sube al servidor.

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

Nota: El código no se traduce, ya que es un bloque de código en Java y debe permanecer en su idioma original para mantener su funcionalidad y claridad.

saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap)

Guarda el mapa de bits en un archivo y lo sube de manera asíncrona.

```
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;
```

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

@Override
protected Void doInBackground(Void... params) {
  try {
    if (baseUrl == null) {
      throw new Exception("baseUrl es nulo");
    }
    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();
  }
}
```

}.execute(); }

Gestión de Fragmentos

La actividad gestiona diferentes fragmentos para manejar los diversos estados de la aplicación, como dil

showDrawFragment(int infoId)
Muestra el fragmento de dibujo.

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

Nota: El código no se traduce, ya que es un bloque de código en Java y debe mantenerse en su idioma original para preservar su funcionalidad y sintaxis.

showWaitFragment()
Muestra un fragmento de espera.

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

showFragment(Fragment fragment)

Reemplaza el fragmento actual con el fragmento especificado.

```
private void showFragment(Fragment fragment) {
  FragmentTransaction trans = getFragmentManager().beginTransaction();
  trans.replace(R.id.rightLayout, fragment);
  trans.commit();
}
```

Manejo de Eventos

Activity maneja diversas interacciones del usuario, como clics en botones y selecciones de menú.

onClick(View v)

Maneja los eventos de clic en diferentes vistas.

```
@Override
public void onClick(View v) {
  int id = v.getId();
  if (id == R.id.drawOk) {
    if (drawView.isDrawFinish()) {
      guardarBitmap();
    } 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) {
    limpiarTodo();
  } 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) {
    irMaterial();
  } else if (id == R.id.upload) {
    com.lzw.commons.Utils.goActivity(cxt, PhotoActivity.class);
  } else if (id == R.id.scale) {
    recortarImagen(curPicUri);
  }
}
```

onActivityResult(int requestCode, int resultCode, Intent data)

Maneja el resultado de otras actividades, como la selección o recorte de imágenes.

```
@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());
    }
```

```
  }
}
```

Funcionalidad de Deshacer y Rehacer

Activity ofrece funcionalidades de deshacer y rehacer para las operaciones de dibujo.

initUndoRedoEnable()
Inicializa las funciones de deshacer y rehacer configurando las funciones de retorno (callbacks).

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

Nota: El código proporcionado está en Java y no requiere traducción, ya que los nombres de métodos y variables deben mantenerse en su idioma original para preservar la funcionalidad del programa.

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

Personalización de DrawView

`DrawView` es una vista personalizada que se utiliza para manejar operaciones de dibujo, eventos táctiles y zoom.

onTouchEvent(MotionEvent event)

Maneja los eventos táctiles para el dibujo y el zoom.

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

Nota: El código no necesita traducción, ya que es un bloque de código en Java y los nombres de métodos y variables deben mantenerse en inglés para mantener la funcionalidad del programa.

```
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();
}
```

```
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;
  }
}

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;
}
```

Traducción al español:

```
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;
}
```

El código permanece igual, ya que es un bloque de código y no debe ser traducido. Sin embargo, si necesitas una explicación en español del funcionamiento del código, aquí está:

Este método `handleMove` se utiliza para manejar el movimiento en la pantalla táctil. Cuando el usuario mueve el dedo, se obtienen las coordenadas `X` e `Y` del evento de movimiento (`MotionEvent`). Si es la primera vez que se detecta un movimiento, se inicializan las coordenadas `lastMoveX` y `lastMoveY` con las coordenadas actuales. Luego, se calcula la distancia recorrida en `X` e `Y` desde la última posición. Si el movimiento excede los límites de la imagen o la pantalla, se ajusta la distancia a 0 para evitar que la imagen se salga de los bordes.

Finalmente, se actualiza el estado a `STATUS_MOVE`, se invalida la vista para redibujar y se actualizan las coordenadas `lastMoveX` y `lastMoveY` con las coordenadas actuales.

```
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();
}
```

onDraw(Canvas canvas)

Dibuja el estado actual de la vista.

```
@Override
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);
  if (scaleMode) {
    switch (status) {
      case STATUS_MOVE:
        mover(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);
    }
  }
}
```

move(Canvas canvas)

Maneja la operación de movimiento durante el escalado.

```
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);
}
```

Nota: El código proporcionado no se traduce, ya que es un bloque de código en Java y debe mantenerse en su idioma original para su correcto funcionamiento.

zoom(Canvas canvas)

Maneja la operación de zoom.

```
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
```

Nota: El código proporcionado está en inglés y no necesita traducción, ya que los nombres de variables, métodos y clases deben mantenerse en su idioma original para garantizar la funcionalidad del programa. Sin embargo, si necesitas una explicación en español sobre lo que hace este código, puedo proporcionarla.

Y = altura - alturaEscalada; } } totalTranslateX = translateX; totalTranslateY = translateY; curBitmapWidth = anchoEscalado; curBitmapHeight = alturaEscalada; matrix.postTranslate(translateX, translateY); canvas.drawBitmap(cacheBm, matrix, null); }

Gestión del Historial

La clase `History` gestiona el historial de dibujos para permitir las funciones de deshacer y rehacer.

saveToStack(Path path, Paint paint)
Guarda la ruta actual y el pincel en la pila.

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

El código anterior permanece igual, ya que es un bloque de código en Java y no debe ser traducido. Sin embargo, si necesitas una explicación en español, aquí está:

Este método `saveToStack` toma un objeto `Path` y un objeto `Paint` como parámetros. Luego, crea una nueva instancia de la clase `Draw`, copia los valores de `path` y `paint` en los atributos correspondientes de la instancia `draw`, y finalmente llama al método `saveToStack` pasando la instancia `draw` como argumento.

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

getBitmapAtDraw(int n)

Devuelve un mapa de bits que representa el estado en un punto dado de la historia.

```java
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;
}
```

undo()

Ejecuta la operación de deshacer.

```
public Bitmap undo() throws UnsupportedOperationException {
  if (canUndo()) {
    curPos--;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("No hay registros para deshacer");
  }
}
```

redo()

Ejecuta la operación de rehacer.

```
public Bitmap redo() throws UnsupportedOperationException {
  if (canRedo()) {
    curPos++;
    if (callBack != null) {
      callBack.onHistoryChanged();
    }
    return getBitmapAtDraw(curPos);
  } else {
    throw new UnsupportedOperationException("No hay registros para rehacer");
  }
}
```

canUndo()

Verifica si es posible deshacer una acción.

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

canRedo()

Verifica si es posible rehacer una acción.

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

El código anterior permanece igual, ya que es un bloque de código en Java y no necesita ser traducido. Sin embargo, si necesitas una explicación en español, aquí está:

Este método `canRedo()` verifica si es posible realizar una acción de "rehacer" (redo). Devuelve `true` si la posición actual (`curPos`) más uno es menor que el tamaño de la lista de historial (`histroy.size()`), lo que significa que hay acciones futuras en el historial que se pueden rehacer. De lo contrario, devuelve `false`.

Conclusión

`DrawActivity` y sus clases relacionadas proporcionan un ejemplo completo de cómo implementar una vista de dibujo personalizada en Android. Este ejemplo demuestra diversas técnicas, como el manejo de eventos táctiles, la gestión del historial de dibujos y la integración con otros componentes, como fragmentos y tareas asíncronas. Al comprender cada componente y algoritmo, puedes aprovechar estas técnicas en tus propias aplicaciones para crear funcionalidades de dibujo potentes e interactivas.