□□□□ Android □□□□□□□

Dieser Blogbeitrag wurde mit Hilfe von ChatGPT-4o verfasst.

Einführung

In diesem Blogbeitrag werden wir die `DrawActivity`-Klasse untersuchen, ein umfassendes Beispiel für die Implementierung einer benutzerdefinierten Zeichenansicht in einer Android-App. Wir werden jede Komponente und die verwendeten Algorithmen aufschlüsseln und detailliert erklären, wie sie zusammenarbeiten, um die gewünschte Funktionalität zu erreichen.

Inhaltsverzeichnis

DrawActivity Übersicht

`DrawActivity` ist die Hauptaktivität, die für die Verarbeitung von Zeichenoperationen, das Zuschneiden von Bildern und die Interaktion mit anderen Komponenten (wie Fragmenten und dem Hochladen von Bildern) zuständig ist. Es bietet eine Benutzeroberfläche, auf der Benutzer zeichnen, rückgängig machen, wiederholen und Bilder bearbeiten können.

```java
public class DrawActivity extends Activity implements View.OnClickListener {
  // Konstanten für Anforderungscodes und Fragment-IDs
  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;

  // Variablen zur Verarbeitung von Bildern und Zeichenoperationen
  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;
}
```

Aktivität initialisieren

Beim Erstellen einer Activity werden verschiedene Initialisierungsvorgänge durchgeführt, wie das Einrichten der Ansicht, das Laden von initialen Bildern und das Konfigurieren von Event-Listenern.

```
@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();
}
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt und die Übersetzung von Code in der Regel nicht sinnvoll ist.

findView()

Diese Methode initialisiert die in der Activity verwendeten Ansichten.

```
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()

Legt die Größe der Zeichenansicht fest.

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

Übersetzung:

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird. Die Methode `setSize` wird verwendet, um die Größe festzulegen, indem zunächst die Größe basierend auf der Ressourcengröße festgelegt wird und dann die Größe der Ansicht (`drawView`) gesetzt wird.

```
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()

Lädt das ursprüngliche Bild, das für die Zeichnung verwendet wird.

```
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);
}
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird.

Bildbearbeitung

Die Activity verarbeitet verschiedene Bildoperationen, wie das Setzen von Bildern über URIs, das Zuschneiden und das Speichern von gezeichneten Bitmaps.

setImageByUri(Uri uri)

Lädt ein Bild von der angegebenen URI und bereitet es für das Zeichnen vor.

```
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)

Startet die Aktivität zum Zuschneiden von Bildern.

```
public void cropIt(Uri uri) {
```

```
  Crop.startPhotoCrop(this, uri, cropPath, CROP_RESULT);
}
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt und die Syntax beibehalten werden sollte.

saveBitmap()
Speichert das gezeichnete Bitmap in einer Datei und lädt es auf den Server hoch.

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt und die Beibehaltung der englischen Begriffe und Syntax für die Funktionalität und Verständlichkeit des Codes wichtig ist.

saveBitmapToFileAndUpload(Bitmap handBitmap, Bitmap originBitmap)
Speichert die Bitmap in einer Datei und lädt sie asynchron hoch.

```
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;
```

In diesem Codeausschnitt wird eine Methode `saveBitmapToFileAndUpload` definiert, die zwei Bitmaps (`handBitmap` und `originBitmap`) als Parameter erhält. Die Methode speichert diese Bitmaps in Dateien und führt anschließend einen asynchronen Task aus. Hier ist eine Übersetzung der Kommentare und Beschreibungen ins Deutsche:

- `final String originPath = PathUtils.getOriginPath();`: Ruft den Pfad für das ursprüngliche Bild ab.

- `BitmapUtils.saveBitmapToPath(originBitmap, originPath);`: Speichert das ursprüngliche Bitmap (`originBitmap`) unter dem angegebenen Pfad (`originPath`).
- `final String handPath = PathUtils.getHandPath();`: Ruft den Pfad für das Hand-Bild ab.
- `BitmapUtils.saveBitmapToPath(handBitmap, handPath);`: Speichert das Hand-Bitmap (`handBitmap`) unter dem angegebenen Pfad (`handPath`).
- `new AsyncTask<Void, Void, Void>() { ... }`: Startet einen asynchronen Task, der im Hintergrund ausgeführt wird.

Der asynchrone Task enthält Variablen `res`, `foreBitmap` und `backBitmap`, die möglicherweise für weitere Verarbeitungsschritte verwendet werden.

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

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

  }.execute();
}
```

Fragmentverwaltung

Die Activity verwaltet verschiedene Fragmente, um die verschiedenen Zustände der Anwendung zu handhaben, wie Zeichnen, Erkennen und Warten.

showDrawFragment(int infoId)

Zeigt das Zeichen-Fragment an.

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird. Die Struktur und die Schlüsselwörter bleiben in der Originalsprache, um die Funktionalität und Lesbarkeit des Codes zu gewährleisten.

showWaitFragment()

Zeigt das Warte-Fragment an.

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt und die Beibehaltung der englischen Begriffe und Syntax für die Funktionalität und Verständlichkeit des Codes wichtig ist.

showFragment(Fragment fragment)

Ersetzt das aktuelle Fragment durch das angegebene Fragment.

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird. Die Struktur und die Schlüsselwörter bleiben in der Originalsprache, um die Funktionalität zu erhalten.

Ereignisbehandlung

Activity verarbeitet verschiedene Benutzerinteraktionen, wie beispielsweise das Klicken auf Schaltflächen und die Auswahl von Menüpunkten.

onClick(View v)

Verarbeitet Klick-Ereignisse für verschiedene Ansichten.

```
@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);
  }
}
```

onActivityResult(int requestCode, int resultCode, Intent data)

Verarbeitet das Ergebnis anderer Aktivitäten, wie z.B. die Auswahl oder das Zuschneiden von Bildern.

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

Rückgängig und Wiederherstellen Funktionen

Activity bietet Funktionen zum Rückgängigmachen und Wiederholen von Zeichenoperationen.

initUndoRedoEnable()

Initialisiert die Rückgängig- und Wiederherstellen-Funktionen durch das Setzen von Rückruffunktionen.

```
void initUndoRedoEnable() {
  drawView.history.setCallBack(new History.CallBack() {
    @Override
    public void onHistoryChanged() {
      setUndoRedoEnable();
```

```
      if (curFragmentId != DRAW_FRAGMENT) {
        showDrawFragment(curDrawMode);
      }
    }
  });
}
```

Übersetzung:

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um Programmcode handelt, der in der Regel in der Originalsprache belassen wird, um die Funktionalität und Lesbarkeit zu gewährleisten.

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

Benutzerdefinierte DrawView

`DrawView` ist eine benutzerdefinierte Ansicht, die für die Handhabung von Zeichenoperationen, Touch-Ereignissen und Zoomfunktionen zuständig ist.

onTouchEvent(MotionEvent event)
Verarbeitet die Touch-Ereignisse für das Zeichnen und Zoomen.

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

Übersetzung:

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird. Der Text wurde jedoch in den deutschen Kontext gestellt.

```
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();
}
```

Übersetzung:

```
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();
}
```

Erklärung:

- `MotionEvent.ACTION_DOWN`: Wird ausgelöst, wenn der Benutzer den Bildschirm berührt.

- `MotionEvent.ACTION_MOVE`: Wird ausgelöst, wenn der Benutzer den Finger über den Bildschirm bewegt.
- `MotionEvent.ACTION_UP`: Wird ausgelöst, wenn der Benutzer den Finger vom Bildschirm hebt.
- `path.moveTo(x, y)`: Setzt den Startpunkt des Pfades auf die Koordinaten (x, y).
- `path.quadTo(preX, preY, x, y)`: Zeichnet eine quadratische Bézier-Kurve von den vorherigen Koordinaten (preX, preY) zu den aktuellen Koordinaten (x, y).
- `Matrix matrix1 = new Matrix(); matrix.invert(matrix1);`: Erstellt eine invertierte Matrix, die verwendet wird, um den Pfad zu transformieren.
- `paint.setStrokeWidth(strokeWidth * 1.0f / totalRatio);`: Setzt die Strichbreite des Pinsels basierend auf dem aktuellen Zoomfaktor.
- `history.saveToStack(path, paint);`: Speichert den aktuellen Pfad und den Pinselzustand im Verlauf.
- `cacheCanvas.drawPath(path, paint);`: Zeichnet den Pfad auf dem Cache-Canvas.
- `paint.setStrokeWidth(strokeWidth);`: Setzt die Strichbreite des Pinsels zurück auf den ursprünglichen Wert.
- `path.reset();`: Setzt den Pfad zurück, um ihn für die nächste Zeichnung vorzubereiten.
- `setPrev(event);`: Speichert die aktuellen Koordinaten für die nächste Bewegung.
- `invalidate();`: Fordert eine erneute Darstellung der View an.

```
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;
  }
}
```

Übersetzung:

```
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;
  }
}
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um Programmcode handelt, der in der Regel in der Originalsprache belassen wird, um die Funktionalität und Lesbarkeit zu gewährleisten.

```
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;
}
```

Übersetzung:

```
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;
}
```

Der Code bleibt auf Englisch, da es sich um Programmcode handelt, der in der Regel nicht übersetzt wird. Die Logik und die Variablennamen bleiben unverändert, um die Funktionalität des Codes beizubehalten.

```
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)

Zeichnet den aktuellen Zustand der Ansicht.

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

move(Canvas canvas)

Verarbeitet die Bewegungsoperation während der Skalierung.

```
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);
}
```

zoom(Canvas canvas)

Behandelt Zoom-Operationen.

```
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
```

Der obige Code ist ein Ausschnitt aus einer Java-Methode, die für das Zoomen eines Bildes auf einem Canvas verantwortlich ist. Da der Code technischer Natur ist und spezifische Begriffe und Konzepte verwendet, die in der Programmierung universell sind, bleibt er in der Regel unverändert, auch in einer deutschen Übersetzung. Der Code selbst wird nicht übersetzt, da dies die Funktionalität beeinträchtigen könnte. Stattdessen wird der umgebende Text, der den Code erklärt, übersetzt.

In diesem Fall ist der Code jedoch unvollständig, da der letzte Teil fehlt. Wenn Sie den vollständigen Code oder weitere Erklärungen benötigen, lassen Sie es mich wissen!

Y = Höhe - skalierteHöhe; } } totalTranslateX = translateX; totalTranslateY = translateY; curBitmapWidth = skalierteBreite; curBitmapHeight = skalierteHöhe; matrix.postTranslate(translateX, translateY); canvas.drawBitmap(cacheBm, matrix, null); }

```
### Verlaufverwaltung

Die `History`-Klasse verwaltet den Zeichnungsverlauf, um Rückgängig- und Wiederherstellen-Funktionen zu

**saveToStack(Path path, Paint paint)**
Speichert den aktuellen Pfad und den Pinsel auf dem Stapel.

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird. Die Struktur und die Schlüsselwörter bleiben in der Originalsprache erhalten, um die Funktionalität und Lesbarkeit des Codes zu gewährleisten.

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

getBitmapAtDraw(int n)
Gibt eine Bitmap zurück, die den Zustand an einem bestimmten Punkt in der Historie darstellt.

```
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()

Führt eine Rückgängig-Operation aus.

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

redo()

Führt den Wiederherstellungsvorgang aus.

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

canUndo()

Überprüft, ob eine Rückgängig-Aktion möglich ist.

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

canRedo()

Überprüft, ob eine Rückgängig-Aktion rückgängig gemacht werden kann.

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

Fazit

`DrawActivity` und die zugehörigen Klassen bieten ein umfassendes Beispiel für die Implementierung einer benutzerdefinierten Zeichenansicht in Android. Es demonstriert verschiedene Techniken, darunter die Handhabung von Touch-Ereignissen, die Verwaltung der Zeichnungshistorie sowie die Integration mit anderen Komponenten wie Fragmenten und asynchronen Aufgaben. Durch das Verständnis jeder Komponente und jedes Algorithmus können Sie diese Techniken in Ihren eigenen Anwendungen nutzen, um leistungsstarke und interaktive Zeichenfunktionen zu erstellen.