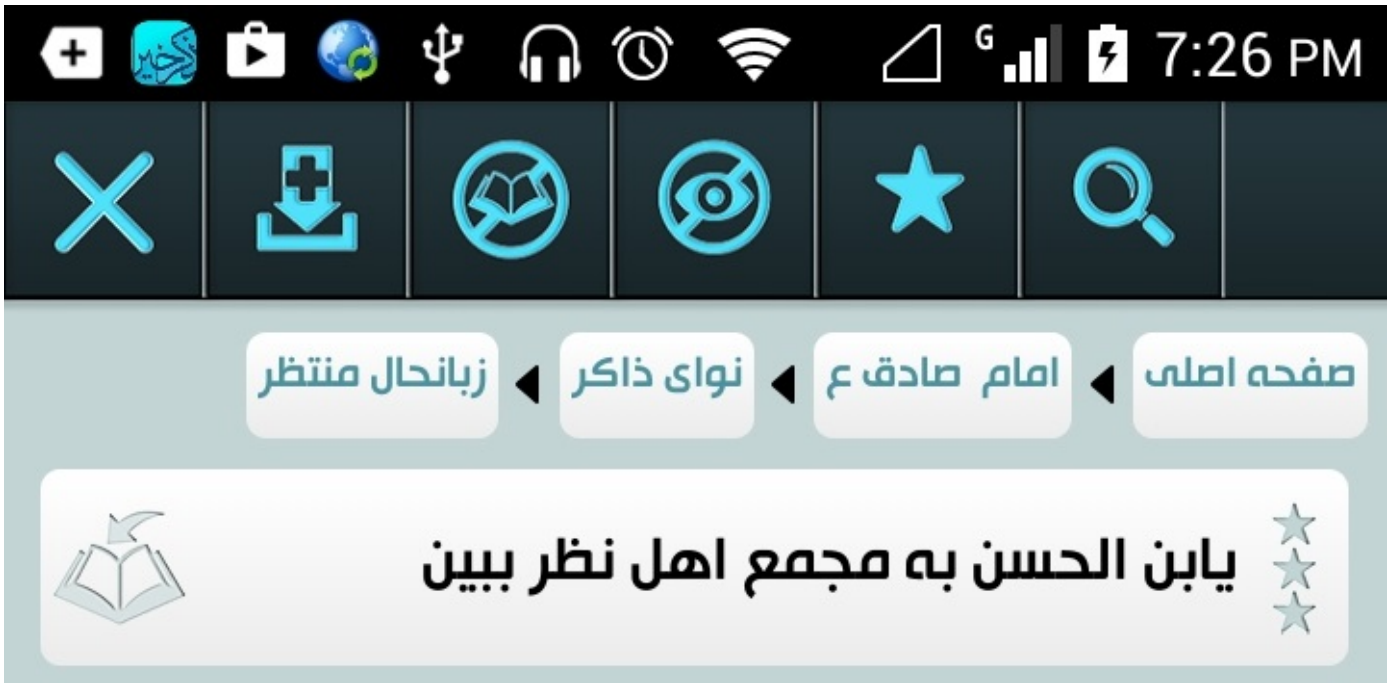
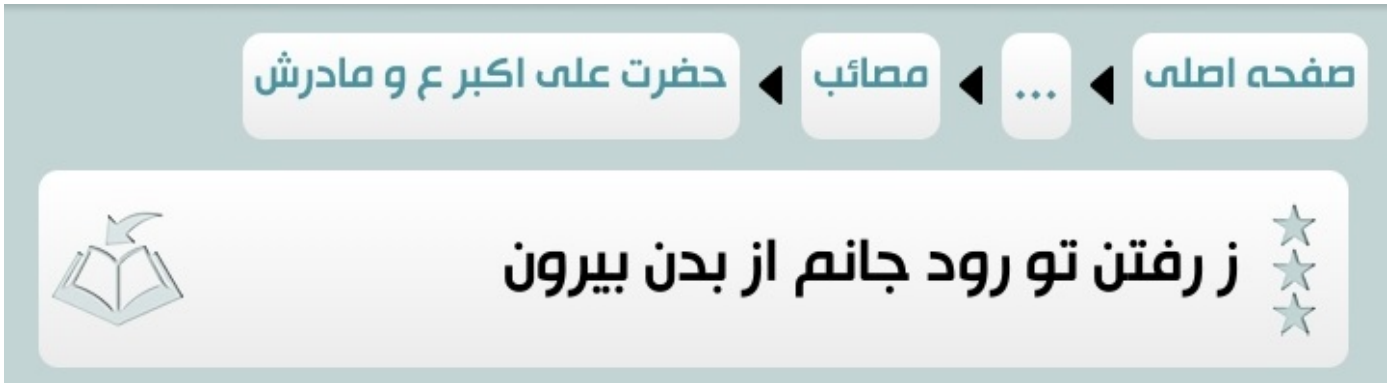
عنوان: **آشنایی با سورس AndroidBreadCrumb**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۸/۰۹ ۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: Android, Java

زمانی که سیستم عامل های GUI مثل ویندوز به بازار آمدند، یکی از قسمت‌های گرافیکی آن‌ها AddressBar نام داشت که مسیر حرکت آن‌ها را در فایل سیستم نشان میداد و در سیستم عامل‌های متنی CLI با دستور cd یا pwd انجام می‌شد. بعدها در وب هم همین حرکت با نام BreadCrumb صورت گرفت که به عنوان مثال مسیر رسیدن به صفحه‌ی یک محصول یا یک مقاله را نشان می‌داد. در یک پروژه‌ی اندرویدی نیاز بود تا یک ساختار درختی را پیاده سازی کنم، ولی در برنامه‌های اندروید ایجاد یک درخت، کار هوشمندانه و مطلوبی نیست و روش کار به این صورت است که یک لیست از گروه‌های والد را نمایش داده و با انتخاب هر آیتم لیست به آیتم‌های فرزند تغییر میکند. حالا مسئله این بود که کاربر باید مسیر حرکت خودش را بشناسد. به همین علت مجبور شدم یک BreadCrumb را برای آن طراحی کنم که در زیر تصویر آن را مشاهده می‌کنید.



از نکات جالب توجه در مورد این ماژول می‌توان گفت که قابلیت این را دارد تا تصمیمات خود را بر اساس اندازه‌های مختلف صفحه نمایش بگیرد. به عنوان مثال اگر آیتم‌های بالا بیشتر از سه عدد باشد و در صفحه جا نشود از یک مسیر جعلی استفاده می‌کند و همه‌ی آیتم‌ها با اندیس شماره 1 تا index-3 را درون یک آیتم با عنوان (...) قرار می‌دهد که من به آن می‌گویم مسیر جعلی. به عنوان نمونه مسیر تصویر بالا در صفحه جا شده است و نیازی به این کار دیده نشده است. ولی تصویر زیر از آن جا که مسیر، طول width صفحه نمایش رد کرده است، نیاز است تا چنین کاری انجام شود. موقعی‌که کاربر آیتم ... را کلیک کند، مسیر باز شده و به محل index-3 حرکت می‌کند. یعنی دو مرحله به عقب باز می‌گردد.



نگاهی به کارکرد ماژول

قبل از توضیح در مورد سورس، اجازه دهید نحوه‌ی استفاده از آن را ببینیم.

این سورس شامل دو کلاس است که ساده‌ترین کلاس آن AndBreadCrumbItem می‌باشد که مشابه کلاس ListItem در بخش وب دات نت است و دو مقدار، یکی متن و دیگری Id را می‌گیرد:

سورس:

```
public class AndBreadCrumbItem {

    private int Id;
    private String diplayText;

    public AndBreadCrumbItem(int Id, String displayText)
    {
        this.Id=Id;
        this.diplayText=displayText;
    }
    public String getDiplayText() {
        return diplayText;
    }
    public void setDiplayText(String diplayText) {
        this.diplayText = diplayText;
    }
    public int getId() {
        return Id;
    }
    public void setId(int id) {
        Id = id;
    }
}
```

به عنوان مثال می‌خواهیم یک breadcrumb را با مشخصات زیر بسازیم:

```
AndBreadCrumbItem itemhome=new AndBreadCrumbItem(0,"Home");
AndBreadCrumbItem itemproducts=new AndBreadCrumbItem(12,"Products");
 AndBreadCrumbItem itemdigital=new AndBreadCrumbItem(15,"Digital");
AndBreadCrumbItem itemhdd=new AndBreadCrumbItem(56,"Hard Disk Drive");
```

حال از کلاس اصلی یعنی AndBreadCrumb استفاده می‌کنیم و آیتم‌ها را به آن اضافه می‌کنیم:

```
AndBreadCrumb breadCrumb=new AndBreadCrumb(this);

        breadCrumb.AddNewItem(itemhome);
        breadCrumb.AddNewItem(itemproducts);
        breadCrumb.AddNewItem(itemdigital);
```

```
        breadCrumb.AddNewItem(itemhdd);
```

به این نکته دقت داشته باشید که با هر شروع مجدد چرخه‌ی Activity، حتما شیء Context این کلاس را به روز نمایید تا در رسم المان‌ها به مشکل برنخورد. می‌توانید از طریق متد زیر context را مقداردهی نمایید:

```
breadCumb.setContext(this);
```

هر چند راه حل پیشنهادی این است که این کلاس را نگهداری ننماید و از یک لیست ایستا جهت نگهداری BreadCrumbItemAndها استفاده کنید تا باهر بار فراخوانی رویدادهای اولیه چون oncreate یا onstart و.. شی BreadCrumb را پر نمایید.

پس از افزودن آیتم ها، تنظیمات زیر را اعمال نمایید:

```
LinearLayout layout=(LinearLayout)getActivity().findViewById(R.id.breadcumblayout);
        layout.setPadding(8, 8, 8, 8);
        breadCrumb.setLayout(layout);
        breadCrumb.SetTinyNextNodeImage(R.drawable.arrow);
        breadCrumb.setTextSize(25);
        breadCrumb.SetViewStyleId(R.drawable.list_item_style);
```

در سه خط اول، یک layout از نوع Linear جهت رسم اشیاء به شیء breadcrumb معرفی می‌شود. سپس در صورت تمایل می‌توانید از یک شیء تصویر گرافیکی کوچک هم استفاده کنید که در تصاویر بالا می‌بینید از تصویر یک فلش جهت دار استفاده شده است تا بین هر المان ایجاد شده از آیتم‌ها قرار بگیرد. سپس در صورت تمایل اندازه‌ی قلم متون را مشخص می‌کنید و در آخر هم متد SetViewStyleId هم برای نسبت دادن یک استایل یا selector و ... استفاده می‌شود.
حال برای رسم آن متد UpdatePath را صدا می‌زنیم:

```
breadCrumb.UpdatePath();
```

الان اگر برنامه اجرا شود باید breadcrumb از چپ به راست رسم گردد. برای استفاده‌های فارسی، راست به چپ می‌توانید از متد زیر استفاده کنید:

```
breadCrumb.setRTL(true);
```

در صورت هر گونه تغییری در تنظیمات، مجددا متد UpdatePath را فراخوانی کنید تا عملیات رسم، با تنظمیات جدید آغاز گردد.

در صورتیکه قصد دارید تنظیمات بیشتری چون رنگ متن، فونت متن و ... را روی هر المان اعمال کنید، از رویداد زیر استفاده کنید:

```
breadCrumb.setOnTextViewUpdate(new ITextViewUpdate() {
            @Override
            public TextView UpdateTextView(Context context, TextView tv) {
                tv.setTextColor(...);
                tv.setTypeface(...);
                return tv;
            }
        });
```

با هر بار ایجاد المان که از نوع TextView است، این رویداد فراخوانی شده و تنظیمات شما را روی آن اجرا می‌کند.
همچنین در صورتیکه می‌خواهید بدانید کاربر بر روی چه عنصری کلیک کرده است، از رویداد زیر استفاده کنید:

```
breadCumb.setOnClickListener(new IClickListener() {
            @Override
            public void onClick(int position, int Id) {
                  //...
            }
```

```
        });
```

کد بالا دو آرگومان را ارسال میکند که اولی position یا اندیس مکانی عنصر کلیک شده را بر می‌گرداند و دومی id هست که با استفاده ازکلاس AndBreadCrumbItem به آن پاس کرده‌اید. هنگام کلیک کاربر روی عنصر مورد نظر، برگشت به عقب به طور خودکار صورت گرفته و عناصر بعد از آن موقعیت، به طور خودکار حذف خواهند شد.

آخرین متد موجود که کمترین استفاده را دارد، متد SetNoResize است. در صورتیکه این متد با True مقداردهی گردد، عملیات تنظیم بر اساس صفحه‌ی نمایش لغو می‌شود. این متد برای زمانی مناسب است که به عنوان مثال شما از یک HorozinalScrollView استفاده کرده باشید. در این حالت layout شما هیچ گاه به پایان نمی‌رسد و بهتر هست عملیات اضافه را لغو کنید.

نگاهی به سورس

کلاس زیر شامل بخش‌های زیر است:
فیلدهای خصوصی

```
//=-=--=-=-=-=-=-=-=-=-=-=- Private Properties -=-=-=-=-=-=-=--=-=-=
    private List<AndBreadCrumbItem> items=null;
    private List<TextView> textViews;
    private int tinyNextNodeImage;
    private int viewStyleId;
    private Context context;
    private boolean RTL;
    private float textSize=20;
    private boolean noResize=false;

    LinearLayout layout;
    IClickListener clickListener;
    ITextViewUpdate textViewUpdate;
    LinearLayout.LayoutParams params ;
```

با نگاهی به نام آن‌ها میتوان حدس زد که برای چه کاری استفاده می‌شوند. به عنوان نمونه از اصلی‌ترین‌ها، متغیر items جهت نگهداری آیتم‌های پاس شده استفاده می‌شود و textviews هم برای نگهداری هر breadcrumb یا همان المان TextView که روی صفحه رسم می‌شود.
اینترفیس‌ها هم با حرف I شروع و برای تعریف رویدادها ایجاد شده‌اند. در ادامه از تعدادی متد get و Set برای مقدار دهی بعضی از فیلدهای خصوصی بالا استفاده شده است:

```
//=-=---=-=-=-=-- Constructor =--=-=-=-=-=--=-=-

    public AndBreadCrumb(Context context)
    {
        this.context=context;
        params = new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT);
    }

    //=-=-=--=--=-=-=-=-=-=-=-=-  Public Properties --=-=-=-=-=-=--=-=-=-=-=-=-=-

    //each category would be added to create path
    public void AddNewItem(AndBreadCrumbItem item)
    {
        if(items==null)
            items=new ArrayList<>();
        items.add(item);
    }

    // if you want a pointer or next node between categories or textviews
    public void SetTinyNextNodeImage(int resId) {this.tinyNextNodeImage=resId;}

    public void SetViewStyleId(int resId) {this.viewStyleId=resId;}
```

```
    public void setTextSize(float textSize) {this.textSize = textSize;}

    public boolean isRTL() {
        return RTL;
    }

    public void setRTL(boolean RTL) {
        this.RTL = RTL;
    }

    public void setLayout(LinearLayout layout) {

        this.layout = layout;
    }

    public void setContext(Context context) {
        this.context = context;
    }

    public boolean isNoResize() {
        return noResize;
    }

    public void setNoResize(boolean noResize) {
        this.noResize = noResize;
    }
```

بعد از آن به متدهای خصوصی می‌رسیم که متد زیر، متد اصلی ما برای ساخت breadcrumb است:

```
//primary method for render objects on layout
    private void DrawPath() {


        //stop here if essentail elements aren't present
        if (items == null) return ;
        if (layout == null) return;
        if (items.size() == 0) return;


//we need to get size of layout,so we use the post method to run this thread when ui is ready
        layout.post(new Runnable() {
            @Override
            public void run() {


                //textviews created here one by one
                int position = 0;
                textViews = new ArrayList<>();
                for (AndBreadCrumbItem item : items) {
                    TextView tv = MakeTextView(position, item.getId());
                    tv.setText(item.getDiplayText());
                    textViews.add(tv);
                    position++;
                }


                //add textviews on layout
                AddTextViewsOnLayout();

                //we dont manage resizing anymore
                if(isNoResize()) return;

                //run this code after textviews Added to get widths of them
                TextView last_tv=textViews.get(textViews.size()-1);
                last_tv.post(new Runnable() {
                    @Override
                    public void run() {
                        //define width of each textview depend on screen width
                        BatchSizeOperation();
                    }
                });

            }
        });


    }
```

متد DrawPath برای ترسیم breadcumb است و می‌توان گفت اصلی‌ترین متد این کلاس است. در سه خط اول، عناصر الزامی را که باید مقداردهی شده باشند، بررسی می‌کند. این موارد وجود آیتم‌ها و layout است. اگر هیچ یک از اینها مقدار دهی نشده باشند، عملیات رسم خاتمه می‌یابد. بعد از آن یک پروسه‌ی UI جدید را در متد post شیء Layout معرفی می‌کنیم. این متد زمانی این پروسه را صدا می‌زند که layout در UI برنامه جا گرفته باشد. دلیل اینکار این است که تا زمانی که ویوها در UI تنظیم نشوند، نمی‌توانند اطلاعاتی چون پهنا و ارتفاع را برگردانند و همیشه مقدار 0 را باز می‌گردانند. پس ما بامتد post اعلام می‌کنیم زمانی این پروسه را اجرا کن که وضعیت UI خود را مشخص کرده‌ای.

به عنوان نمونه کد زیر را ببینید:

```
TextView tv=new TextView(this);
tv.getWidth(); //return 0
layout.add(tv);
tv.getWidth(); //return 0
```

در این حالت کنترل در هر صورتی عدد ۰ را به شما باز می‌گرداند و نمی‌توانید اندازه‌ی آن را بگیرید مگر اینکه درخواست یک callback بعد از رسم را داشته باشید که این کار از طریق متد post انجام می‌گیرد:

```
TextView tv=new TextView(this);
tv.post(new Runnable() {
                    @Override
                    public void run() {
                        tv.getWidth(); //return x
                    }
                });
```

در اینجا مقدار واقعی x بازگردانده می‌شود.

باز می‌گردیم به متد DrawPath و داخل متد post
در اولین خط این پروسه به ازای هر آیتم، یک TextView توسط متد MakeTextView ساخته می‌شود که شامل کد زیر است:

```
private TextView MakeTextView(final int position, final int Id)
    {
        //settings for cumbs
        TextView tv=new TextView(this.context);
        tv.setEllipsize(TextUtils.TruncateAt.END);
        tv.setSingleLine(true);
        tv.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
        tv.setBackgroundResource(viewStyleId);

        /*call custom event - this event will be fired when user click on one of
         textviews and returns position of textview and value that user sat as id
         */
        tv.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                SetPosition(position);
                clickListener.onClick(position, Id);
            }
        });

        //if user wants to update each textviews
        if(textViewUpdate!=null)
            tv=textViewUpdate.UpdateTextView(context,tv);

        if(isRTL())
            tv.setRotationY(180);

        return tv;
    }
```

در خطوط اولیه، یک Textview ساخته و متد Ellipsize را با Truncate.END مقداردهی می‌نماید. این مقدار دهی باعث می‌شود

اگر متن، در Textview جا نشد، ادامه‌ی آن با ... مشخص شود. در خط بعدی Textview را تک خطه معرفی می‌کنیم. در خط بعدی اندازه‌ی قلم را بر اساس آنچه کاربر مشخص کرده است، تغییر می‌دهیم و بعد هم استایل را برای آن مقداردهی می‌کنیم. بعد از آن رویداد کلیک را برای آن مشخص می‌کنیم تا اگر کاربر بر روی آن کلیک کرد، رویداد اختصاصی خودمان را فراخوانی کنیم.
در خط بعدی اگر rtl با true مقدار دهی شده باشد، textview را حول محور Y چرخش می‌دهد تا برای زبان‌های راست به چپ چون فارسی آماده گردد و در نهایت Textview ساخته شده و به سمت متد DrawPath باز می‌گرداند.

بعد از ساخته شدن TextViewها، وقت آن است که به Layout اضافه شوند که وظیفه‌ی اینکار بر عهده‌ی متد AddTextViewOnLayout است:

```
//this method calling by everywhere to needs add textviews on the layout like master method :drawpath
    private void AddTextViewsOnLayout()
    {
        //prepare layout
        //remove everything on layout for recreate it
        layout.removeAllViews();
        layout.setOrientation(LinearLayout.HORIZONTAL);
        layout.setVerticalGravity(Gravity.CENTER_VERTICAL);
        if(isRTL())
            layout.setRotationY(180);



        //add textviews one by one

        int position=0;
        for (TextView tv:textViews)
        {
            layout.addView(tv,params);

            //add next node image between textviews if user defined a next node image
            if(tinyNextNodeImage>0)
                if(position<(textViews.size()-1)) {
                    layout.addView(GetNodeImage(), params);
                    position++;
                }
        }

    }
```

در چند خط اول، Layout آماده سازی می‌شود. این آماده سازی شامل پاکسازی اولیه Layout یا خالی کردن ویوهای درون آن است که می‌تواند از رندر قبلی باشد. افقی بودن جهت چینش Layout، در مرکز نگاه داشتن ویوها و نهایتا چرخش حول محور Y در صورت true بودن خاصیت RTL است. در خطوط بعدی یک حلقه وجود دارد که Textviewهای ایجاد شده را یک به یک در Layout می‌چیند و اگر کاربر تصویر گرافیکی را هم به (همان فلش‌های اشاره‌گر) متغیر tinyNextNodeImage نسبت داده باشد، آن‌ها را هم بین TextViewها می‌چیند و بعد از پایان یافتن کار، مجددا به متد DrawPath باز می‌گردد.
تا به اینجا کار چیدمان به ترتیب انجام شده است ولی از آنجا که اندازه‌ی Layout در هر گوشی و در دو حالت حالت افقی یا عمودی نگه داشتن گوشی متفاوت است، نمی‌توان به این چینش اعتماد کرد که به چه نحوی عناصر نمایش داده خواهند شد و این مشکل توسط متد BatchSizeOperation (تغییر اندازه دسته جمعی) حل می‌گردد. در اینجا هم باز متد post به آخرین textview اضافه شده است. به این علت که موقعی‌که همه‌ی textviewها در ui جا خوش کردند، بتوانیم به خاصیت‌های ui آن‌ها دستیابی داشته باشیم. حالا بعد از ترسیم باید اندازه آن‌ها را اصلاح کنیم. قدم به قدم متد BatchSizeOperation را بررسی می‌کنیم:

```
//set textview width depend on screen width
private void BatchSizeOperation()
{
//get width of next node between cumbs
Bitmap tinyBmap = BitmapFactory.decodeResource(context.getResources(), tinyNextNodeImage);
int tinysize=tinyBmap.getWidth();
//get sum of nodes
tinysize*=(textViews.size()-1);
...
}
```

ابتدا لازم است طول مسیری که همه ویوها یا المان‌های ما را دارند، به دست آوریم. اول از تصویر کوچک شروع می‌کنیم و پهنای آن را می‌گیریم. سپس عدد به دست آمده را در تعداد آن ضرب می‌کنیم تا جمع پهناها را داشته باشیم. سپس نوبت به TextViewها می‌رسد.

```
//get width size of screen(layout is screen here)
        int screenWidth=GetLayoutWidthSize();

        //get sum of arrows and cumbs width
        int sumtvs=tinysize;
        for (TextView tv : textViews) {

            int width=tv.getWidth();
            sumtvs += width;
        }
```

در ادامه‌ی این متد، متد GetLayoutWidthSize را صدا می‌زنیم که وظیفه‌ی آن برگرداندن پهنای layout است و کد آن به شرح زیر است:

```
private int GetLayoutWidthSize()
    {
        int width=layout.getWidth();
        int padding=layout.getPaddingLeft()+layout.getPaddingRight();
        width-=padding;
        return width;
    }
```

در این متد پهنا به احتساب paddingهای چپ و راست به دست می‌آید و مقدار آن را به عنوان اندازه‌ی صفحه نمایش، تحویل متد والد می‌دهد. در ادامه هم پهنای هر Textview محاسبه شده و جمع کل آن‌ها را با اندازه‌ی صفحه مقایسه می‌کند. اگر کوچکتر بود، کار این متد در اینجا تمام می‌شود و نیازی به تغییر اندازه نیست. ولی اگر نبود کد ادامه می‌یابد:

```
private void  BatchSizeOperation()
    {
        ....

    //if sum of cumbs is less than screen size the state is good so return same old textviews
        if(sumtvs<screenWidth)
            return ;


        if(textViews.size()>3)
        {
            //make fake path
            MakeFakePath();

            //clear layout and add textviews again
            AddTextViewsOnLayout();
        }

        //get free space without next nodes -> and spilt rest of space to textviews count to get space
for each textview
        int freespace =screenWidth-tinysize;
        int each_width=freespace/textViews.size();

        //some elements have less than each_width,so we should leave size them and calculate more space
again
        int view_count=0;
        for (TextView tv:textViews)
        {
            if (tv.getWidth()<=each_width)
                freespace=freespace-tv.getWidth();
            else
                view_count++;
        }
        if (view_count==0) return;

        each_width=freespace/view_count;
        for (TextView tv:textViews)
        {
            if (tv.getWidth()>each_width)
```

```
                tv.setWidth(each_width);
        }


    }
```

اگر آیتم‌ها بیشتر از سه عدد باشند، می‌توانیم از حالت مسیر جعلی استفاده کنیم که توسط متد MakeFakePath انجام می‌شود. البته بعد از آن هم باید دوباره viewها را چینش کنیم تا مسیر جدید ترسیم گردد، چون ممکن است بعد از آن باز هم جا نباشد یا آیتم‌ها بیشتر از سه عدد نیستند. در این حالت، حداقل کاری که می‌توانیم انجام دهیم این است که فضای موجود را بین آن‌ها تقسیم کنیم تا همه‌ی کاسه، کوزه‌ها سر آیتم آخر نشکند و متنش به ... تغییر یابد و حداقل از هر آیتم، مقداری از متن اصلی نمایش داده شود. پس میانگین فضای موجود را گرفته و بر تعداد المان‌ها تقسیم می‌کنیم. البته این را هم باید در نظر گرفت که در تقسیم بندی، بعضی آیتم‌ها آن مقدار پهنا را نیاز ندارند و با پهنای کمتر هم می‌شود کل متنشان را نشان داد. پس یک کار اضافه‌تر این است که مقدار پهنای اضافی آن‌ها را هم حساب کنیم و فقط آیتم‌هایی را پهنا دهیم که به مقدار بیشتری از این میانگین احتیاج دارند. در اینجا کار به پایان می‌رسد و مسیر نمایش داده می‌شود.

نحوه‌ی کارکرد متد MakeFakePath بدین صورت است که 4 عدد TextView را ایجاد کرده که المان‌های با اندیس 0 و 2 و 3 به صورت نرمال و عادی ایجاد شده و همان کارکرد سابق را دارند. ولی المان شماره دو با اندیس 1 با متن ... نماینده‌ی آیتم‌های میانی است و رویدادکلیک آن به شکل زیر تحریف یافته است:

```
//if elements are so much(mor than 3),we make a fake path to decrease elements
    private void MakeFakePath()
    {
        //we make 4 new elements that index 1 is fake element and has a rest of real path in its heart
        //when user click on it,path would be opened
        textViews=new ArrayList<>(4);
        TextView[] tvs=new TextView[4];
        int[] positions= {0,items.size()-3,items.size()-2,items.size()-1};

        for (int i=0;i<4;i++)
        {
            //request for new textviews
            tvs[i]=MakeTextView(positions[i],items.get(positions[i]).getId());

            if(i!=1)
                tvs[i].setText(items.get(positions[i]).getDiplayText());
            else {
                tvs[i].setText("...");
                //override click event and change it to part of code to open real path by call
setposition method and redraw path
                tvs[i].setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        int pos = items.size() - 3;
                        int id = items.get(pos).getId();
                        SetPosition(items.size() - 3);
                        clickListener.onClick(pos, id);
                    }
                });
            }
            textViews.add(tvs[i]);
        }
    }
```

این رویداد با استفاده از setPosition به آیتم index-3 بازگشته و مجددا المان‌ها رسم می‌گردند و سپس رویداد کلیک این آیتم را هم اجرا می‌کند و المان‌های با اندیس 2 و 3 را به ترتیب به رویدادهای index-1 و index-2 متصل می‌کنیم.