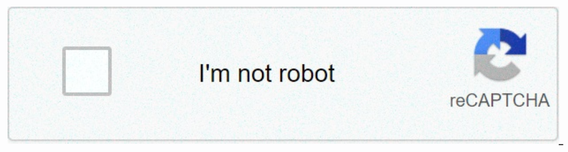
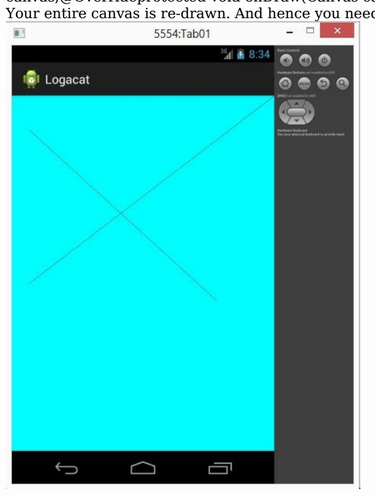


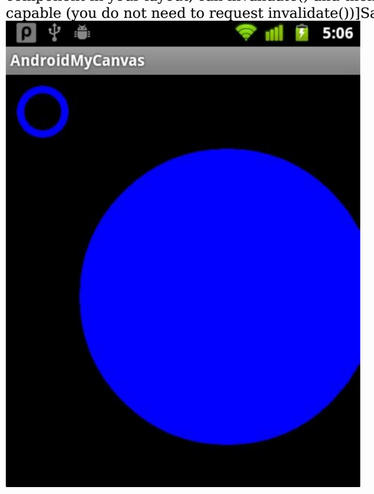
Continue

Draw dotted line android canvas

Burn Ignorance is a knowledge initiative by Mindfire Solutions© Copyright Burn Ignorance 2007-2016. The setLineDash() method of the Canvas 2D API's CanvasRenderingContext2D interface sets the line dash pattern used when stroking lines. It uses an array of values that specify alternating lengths of lines and gaps which describe the pattern. Note: To return to using solid lines, set the line dash list to an empty array. segments An Array of numbers that specify distances to alternately draw a line and a gap (in coordinate space units). If the number of elements in the array is odd, the elements of the array get copied and concatenated. For example, [5, 15, 25] will become [5, 15, 25, 5, 15, 25]. If the array is empty, the line dash list is cleared and line strokes return to being solid. This example uses the setLineDash() method to draw a dashed line above a solid line. HTML JavaScript const canvas = document.getElementById("canvas"); const ctx = canvas.getContext("2d"); ctx.beginPath(); ctx.setLineDash([5, 15]); ctx.moveTo(0, 50); ctx.lineTo(300, 50); ctx.stroke(); ctx.beginPath(); ctx.setLineDash([]); ctx.moveTo(0, 100); ctx.lineTo(300, 100); ctx.stroke(); ResultThis example illustrates a variety of common line dash patterns. HTML JavaScript The drawDashedLine() function created below makes the drawing of multiple dashed lines simple. It receives a pattern array as its only parameter. function drawDashedLine(pattern) { ctx.beginPath(); ctx.setLineDash(pattern); ctx.moveTo(0, y); ctx.lineTo(300, y); ctx.stroke(); y += 20; } const canvas = document.getElementById("canvas"); const ctx = canvas.getContext("2d"); let y = 15; drawDashedLine([]); drawDashedLine([1, 1]); drawDashedLine([10, 10]); drawDashedLine([20, 5]); drawDashedLine([15, 3, 3, 3]); drawDashedLine([20, 3, 3, 3, 3, 3, 3, 3]); drawDashedLine([12, 3, 3]); ResultSpecificationHTML Standard # dom-context-2d-setlinedash-devBCD tables only load in the browser Would you like toCreate your own drawing (UI) on the screen OR create custom views ?Modify existing views and customize their look and feel ?Draw any shape, view or just bitmaps ?Create something which isn't already available ?The power of Android for free hand drawing on pen and paper !Android Canvas gives you exactly that. Just dive in and create your own magic.If you know the basics and directly want to view the code, find the entire source code here on GitHub.So what exactly is Android Canvas ?The Android framework APIs provides a set of 2D drawing APIs that allow you to render your own custom graphics onto a canvas or to modify existing Views to customize their look and feel. Basically, Canvas is a class in Android that performs 2D drawing onto the screen of different objects.Your mobile screen is your canvasJust consider you mobile screen as a blank paper and draw on it. You need to define anchor points and shapes so as to draw on the screen. Remember the school level graphs ? Something very similar.Define X & Y coordinates and the shape you want.Create your own custom view classJust create custom view class. Since you want to draw your own UI , extend View class to get the lifecycle of the basic view hierarchy.public class CustomView extends View { public CustomView(Context context, AttributeSet attrs) { super(context, attrs); }}Define a paint object with default colors and styling// defines paint and canvasprivate Paint drawPaint;// Setup paint with color and stroke stylesprivate void setupPaint() { drawPaint = new Paint(); drawPaint.setColor(Color.BLUE); drawPaint.setAntiAlias(true); drawPaint.setStrokeWidth(5); drawPaint.setStyle(Paint.Style.FILL_AND_STROKE); drawPaint.setStrokeJoin(Paint.Join.ROUND); drawPaint.setStrokeCap(Paint.Cap.ROUND);}Create and initialise the paint object in your constructor only. mole ratio worksheet Most of the times, our basic settings don't change. We can then use this paint object every where else in the code and only change properties we want.public CustomView(Context context, @Nullable AttributeSet attrs) { super(context, attrs); setupPaint();}The magic methods : onDraw() and invalidate()All of canvas drawing will happen in onDraw method. Whenever you want to draw any custom objects , you set the paint styling, call default draw.. API methods. All these internally call onDraw.Get your canvas instance in onDraw and save it for drawing.private Canvas canvas;@Overrideprotected void onDraw(Canvas canvas) { super.onDraw(canvas); this.canvas = canvas;Every time, you draw something new on the canvas , you need to refresh it.
Your entire canvas is re-drawn. And hence you need to perform minimal operations in onDraw().To tell the view, that is has to refresh use invalidate() method.canvas.invalidate();Remember our paint object is initialised in constructor so that we don't create it again and again on draw. distributive property of multiplication worksheets 6th grade

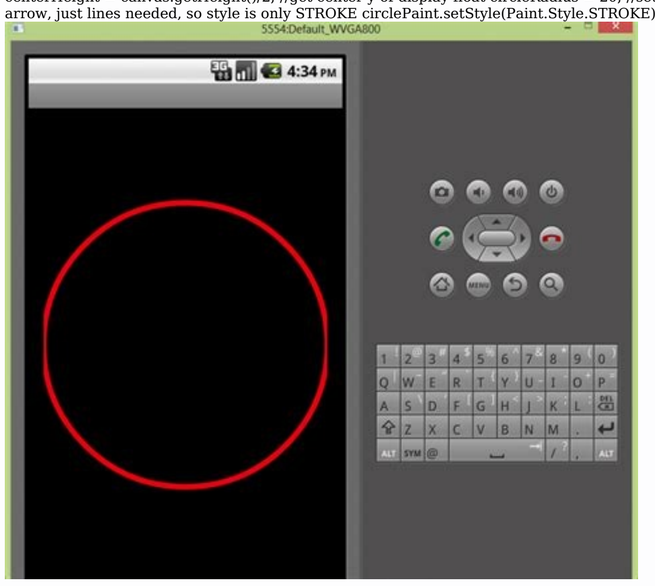


OnDraw gets called every single time you want to change anything on the UI. So it's an expensive call. We don't want to do anything extra than required on onDraw method.Drawing basicsA variety of basic draw API's are available on the canvas object. grundfos_pressure_pumps_manual.pdf
We can use these basic API;s to create our own custom shapes and figures. some common ones are :drawCircledrawLinedrawOvaldrawPointsdrawTextdrawRectdrawPathDraw LineYou define the two points with their x, y coordinates and draw path between them.Path path = new Path();path.moveTo(x1, y1); path.lineTo(x2, y2); path.close();canvas.drawPath(path, drawPaint);Draw CircleThe simplest shape. tigisuzugoverabukuwa.pdf You just need to specify the x coordinate, y coordinate on the screen and the radius. Also set any paint color if you want.canvas.drawCircle(xCordinate, yCordinate, RADIUS, drawPaint);Draw RectangleCreate a rectangle with x, y, height, width.public void drawRectangle(int x, int y) { drawPaint.setColor(Color.RED); Rect rectangle = new Rect((int) (x - ((0.8) * RADIUS)), (int) (y - ((0.6) * RADIUS)), (int) (x + ((0.8) * RADIUS)), (int) (y + ((0.6 * RADIUS)))); canvas.drawRect(rectangle, drawPaint);}Draw SquareCreate a rectangle object, with the required coordinates, with the same width and height.double squareSideHalf = 1 / Math.sqrt(2);//Consider pivot x,y as centroid.public void drawRectangle(int x, int y) { drawPaint.setColor(Color.RED); Rect rectangle = new Rect((int) (x - (squareSideHalf * RADIUS)), (int) (y - (squareSideHalf * RADIUS)), (int) (x + (squareSideHalf * RADIUS)), (int) (y + ((squareSideHalf * RADIUS)))); canvas.drawRect(rectangle, drawPaint);}Getting tougher : Draw Triangle ()Triangle is basically three vertices connected with a line. You need to find those three vertices and draw a line between them.Below we draw an equilateral triangle/*Select three vertices of triangle. Draw 3 lines between them to form a triangle */public void drawTriangle(int x, int y, int width) { drawPaint.setColor(Color.GREEN); int halfWidth = width / 2; Path path = new Path(); path.moveTo(x, y - halfWidth); // Top path.lineTo(x - halfWidth, y + halfWidth); // Bottom left path.lineTo(x + halfWidth, y + halfWidth); // Bottom right path.lineTo(x, y - halfWidth); // Back to Top path.close(); canvas.drawPath(path, drawPaint);}Update CanvasIf you follow the MVP / MVVM / etc other architectural pattern, you might want to refresh your canvas from other layers. Just get the canvas object , do all your business logic for drawing, and then run invalidate.private void upDateCanvas(Shape shape) { //your business logic for shapes creation etc here canvas.invalidate();}View or SurfaceView ?If you want to know more about multi-threading[Use View : If your application does not require a significant amount of processing or frame-rate speed (perhaps for a chess game, a snake game, or another slowly-animated application). In the same thread as your UI Activity, wherein you create a custom View component in your layout, call invalidate() and then handle the onDraw() callback.Use SurfaceView — If you have high computation or so the application doesn't to wait until the system's View hierarchy is ready to draw and want to run in a separate thread, wherein you manage a SurfaceView and perform draws to the Canvas as fast as your thread is capable (you do not need to request invalidate())]Sample code :You can deep dive into the code here on GitHub and check out details there.Draw different shapes on canvas. Full source code available here on githubIn the next article, we will learn more on handling touch events on Canvas like touch, click, long press, etc.Thats' it.



Thank you for reading.

Please let me know what you liked in the article and what would you like to know more.Full source code available here on GitHubIf you liked the article, show some love by clicking on the button . It will motivate me to write more articles like this. Please share any feedback or tweet about it here. Read my other blog posts here :Happy Coding ! :) Declare this method inside onDraw method: private void drawOvalAndArrow(Canvas canvas){ Paint circlePaint = new Paint(); circlePaint.setStyle(Paint.Style.FILL_AND_STROKE); circlePaint.setAntiAlias(true); circlePaint.setStrokeWidth(2); circlePaint.setColor(Color.CYAN); float centerWidth = canvas.getWidth()/2; //get center x of display float centerHeight = canvas.getHeight()/2; //get center y of display float circleRadius = 20; //set radius float circleDistance = 200; //set distance between both circles //draw circles canvas.drawCircle(centerWidth, centerHeight, circleRadius, circlePaint); canvas.drawCircle(centerWidth+circleDistance, centerHeight, circleRadius, circlePaint); //to draw an arrow, just lines needed, so style is only STROKE circlePaint.setStyle(Paint.Style.STROKE); circlePaint.setColor(Color.RED); //create a path to draw on Path arrowPath = new Path(); //create an invisible oval.



the oval is for "behind the scenes" ,to set the path´ //area. Imagine this is an egg behind your circles. the circles are in the middle of this egg final RectF arrowOval = new RectF(); arrowOval.set(centerWidth, centerHeight-80, centerWidth + circleDistance, centerHeight+80); //add the oval to path arrowPath.addArc(arrowOval,-180,180); //draw path on canvas canvas.drawPath(arrowPath, circlePaint); //draw arrowhead on path start arrowPath.moveTo(centerWidth,centerHeight); //move to the center of first circle arrowPath.lineTo(centerWidth-circleRadius, centerHeight-circleRadius);//draw the first arrowhead line to the left arrowPath.moveTo(centerWidth,centerHeight);//move back to the center arrowPath.lineTo(centerWidth+circleRadius, centerHeight-circleRadius);//draw the next arrowhead line to the right //same as above on path end arrowPath.moveTo(centerWidth+circleDistance,centerHeight); arrowPath.lineTo((centerWidth+circleDistance)-circleRadius, centerHeight-circleRadius); arrowPath.moveTo(centerWidth+circleDistance,centerHeight); arrowPath.lineTo((centerWidth+circleDistance)+circleRadius, centerHeight-circleRadius); //draw the path canvas.drawPath(arrowPath,circlePaint); } Also this will find the two sides of the screen (Landscape mode) and will draw a perfect curve across the screen protected void onDraw(Canvas canvas) { super.onDraw(canvas); PointF mPoint1 = new PointF(w/1.2F, h/1.2F); PointF mPoint2 = new PointF(w/24, h/1.2F); Path myPath1 = new Path(); Paint paint = new Paint(); paint.setAntiAlias(true); paint.setStyle(Style.STROKE); paint.setStrokeWidth(2); paint.setColor(Color.WHITE); myPath1 = drawCurve(canvas, paint, mPoint1, mPoint2); canvas.drawPath(myPath1, paint); } private Path drawCurve(Canvas canvas, Paint paint, PointF mPointa, PointF mPointb) { Path myPath = new Path(); myPath.moveTo(63*w/64, h/10); myPath.quadTo(mPointa.x, mPointa.y, mPointb.x, mPointb.y); return myPath; } Useful references on painting in android: How to draw Arcs in Android using canvas? Basic Painting with Views