2D Graphic (II)

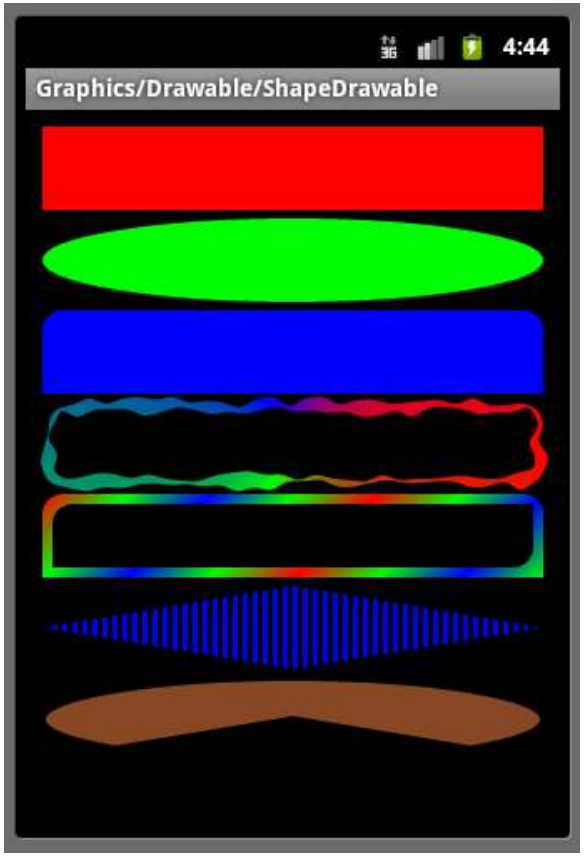
Mobile Software
2019 Fall

What to do next?

- **도형 객체**
- 애니메이션 기초
  - Property animation
  - View animation
  - Drawable animation
- SurfaceView

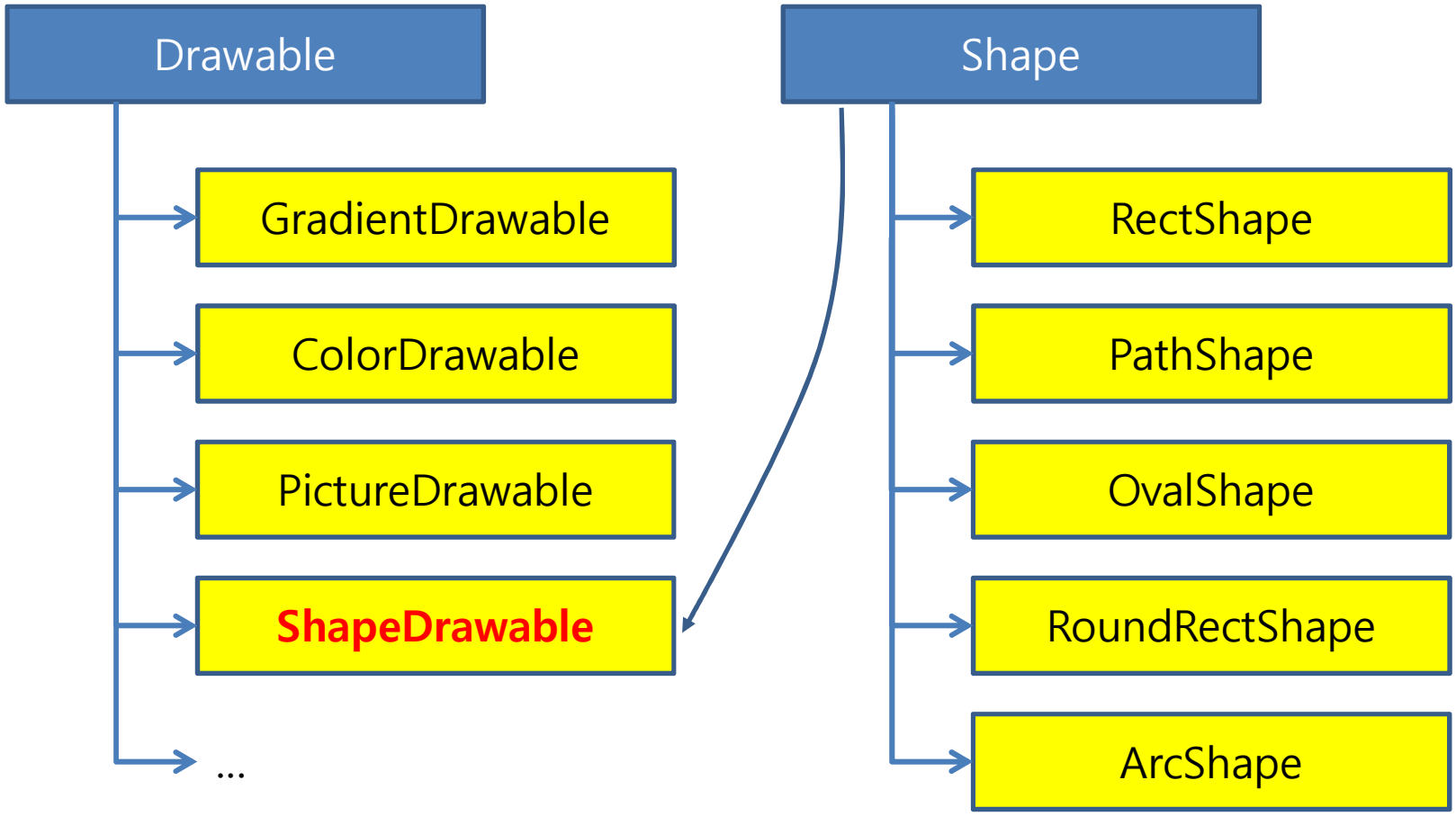
Drawable (도형 객체)

- 사각형이나 원과 같은 2차원 도형을 객체로 정의



- 도형 객체 생성 방법
  - XML 파일
    - 파일 크기를 줄일 수 있음.
    - 모양을 쉽게 바꿀 수 있음.
  - 소스 코드
    - 실행 중에 도형 변경 가능

도형 객체


Drawable
Shape
GradientDrawable
ColorDrawable
PictureDrawable
ShapeDrawable
...
RectShape
PathShape
OvalShape
RoundRectShape
ArcShape

XML로 도형 객체 정의

```
<?xml version="1.0" encoding="utf-8"?>

<shape
   xmlns: android="http://schemas.android.com/apk/res/android"
   android: shape=["rectangle" | "oval" | "line" | "ring"] >
   <corners  .... />
   <gradient .... />
   <padding .... />
   <size  .... />
   <solid  android: color="color" />
   <stroke   android: width="integer" .... />
</shape>
```

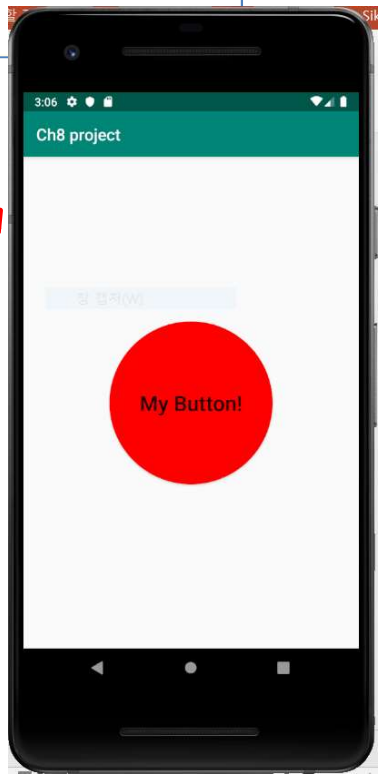
실습 13(a): XML로 도형 객체 정의

res > drawable > 오른쪽 버튼 > New > drawable resource file >"oval.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid android:color="#ff0000" />
</shape>
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="@drawable/oval"
        android:text="My Button"
        android:textSize="20sp" />
</LinearLayout>
```

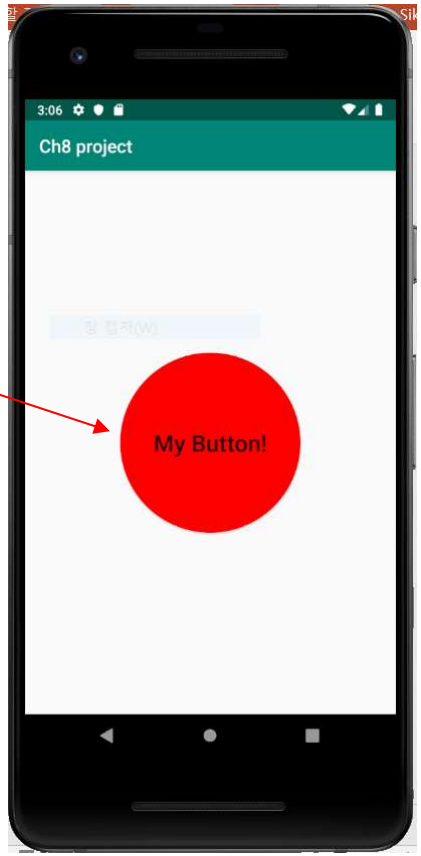


소스 코드 - 1쪽

실습 13(b): 코드에서 도형 정의

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val linearLayout = LinearLayout(this)
        val oval = ShapeDrawable(OvalShape())

        val px: Int = convertUnitToPixel(200f, TypedValue.COMPLEX_UNIT_DIP)
        oval.intrinsicHeight = px
        oval.intrinsicWidth = px
        oval.paint.color = Color.RED

        val button = Button(this)
        button.background = oval
        button.text = "My Button"

        val spx: Int = convertUnitToPixel(10f, TypedValue.COMPLEX_UNIT_SP)
        button.textSize = spx.toFloat()

        linearLayout.addView(button)
        setContentView(linearLayout)
    }

    private fun convertUnitToPixel(value: Float, unit: Int) =
        TypedValue.applyDimension(
            unit, value, resources.displayMetrics
        ).toInt()
}
```



소스 코드 - 2~3쪽

실습 14: Button 배경 지정(1/2)

res > drawable > 오른쪽 버튼 > New > drawable resource file >"myshape.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="14dp" />
    <gradient
        android:angle="45"
        android:centerColor="#7995A8"
        android:centerX="35%"
        android:startColor="#e8e8e8"
        android:endColor="#000000"
        android:type="linear" />
    <padding
        android:bottom="5dp"
        android:left="5dp"
        android:right="5dp"
        android:top="5dp" />
    <size
        android:height="60dp"
        android:width="270dp" />
    <stroke
        android:width="3dp"
        android:color="#878787" />
</shape>
```

소스 코드 - 4쪽

잠깐! class GradientDrawable

```
<gradient
    android:angle="45"
    android:centerColor="#7995A8"
    android:centerX="35%"
    android:startColor="#e8e8e8"
    android:endColor="#000000"
    android:type="linear" />
```

45의 배수이며, (0, 315) 사이의 값

android:angle	Angle of the gradient, used only with linear gradient.
android:bottom	Amount of bottom padding inside the gradient shape.
android:centerColor	Optional center color.
android:centerX	X-position of the center point of the gradient within the shape as a fraction of the width.
android:centerY	Y-position of the center point of the gradient within the shape as a fraction of the height.

The default value is 0.5.
A fractional value, which is a floating point number appended with either % or %p.

실습 14: Button 배경 지정(2/2)

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:text="Button"
        android:textSize="30sp"
        android:textColor="#ffffff"
        android:layout_width="270dp"
        android:layout_height="60dp"
        android:background="@drawable/myshape" />
</LinearLayout>
```

소스 코드 - 4쪽

What to do next?

- 도형 객체
- **애니메이션 기초**
  - **Frame animation**
  - **Tween animation**
  - **Property animation**
- SurfaceView

Animation

- **Frame animation**
  - Drawable animation이라고도 함
    - 주기적으로 이미지(=장면)를 바꿈
    - 여러 장의 이미지를 빠르게 교체하면 움직이는 것처럼 보임.
- **Tween animation**
  - View animation이라고도 함
    - 처음 장면(또는 마지막 장면)을 지정하고
      - 계산을 통해 중간 장면을 연속적으로 생성하는 방식
- **Property animation**
  - 모든 객체의 속성을 애니메이션 할 수 있음
    - **Android SDK 3.0이상**

Frame Animation

- **Drawable** animation
- 영화 필름처럼 여러 개 이미지가 순서대로 재생되어 움직이는 것처럼 보이는 고전 애니메이션

실습 15: XML 파일에서 frame 동작 정의

- XML 파일 → animation을 구성하는 프레임들을 정의
  - **res/ drawable** 폴더에 **rocket.xml** 파일 생성

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item
        android:drawable="@drawable/rocket1"
        android:duration="300" />
    <item
        android:drawable="@drawable/rocket2"
        android:duration="300" />
    <item
        android:drawable="@drawable/rocket3"
        android:duration="300" />
</animation-list>
```

300 → 300ms → 0.3초

소스 코드 - 6쪽

프레임 동작 순서
rocket1 → 0.3초 → rocket2 → 0.3초 → …

oneshot="false" → 무한 반복 실행
oneshot="true" → 한 번 실행하고 멈춤

실습 15: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:layout_margin="5dp"
        android:id="@+id/startBtn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start animation" />
    <Button
        android:layout_margin="5dp"
        android:id="@+id/stopBtn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop animation" />

    <ImageView
        android:id="@+id/rocket_image"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/rocket"/>
</LinearLayout>
```

소스 코드 – 6~7쪽

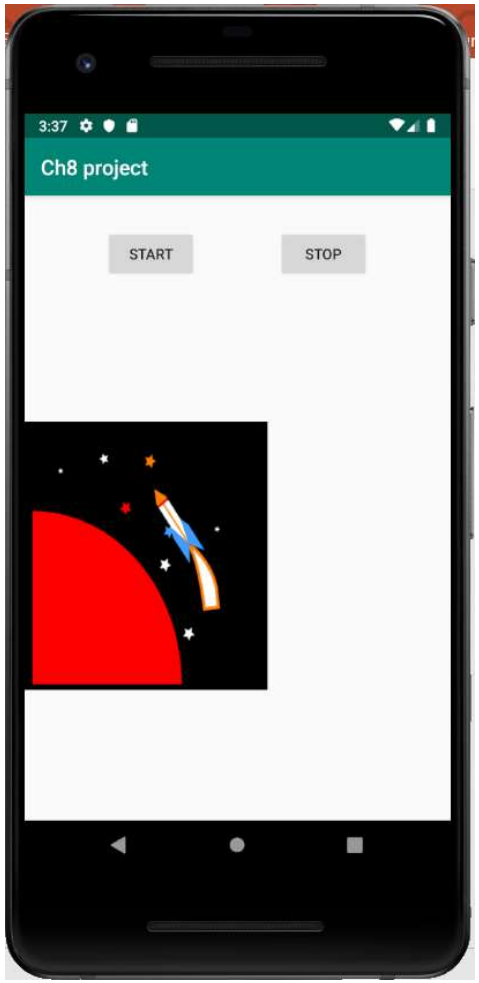
소스 코드에서 직접 지정할 수 있음.

```
rocket_image.setBackgroundResource(R.drawable.rocket)
```

실습 15: Frame Animation

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val rocketAnim = rocketImage.background
                as AnimationDrawable
        startBtn.setOnClickListener {
            rocketAnim.start()
        }
        stopBtn.setOnClickListener {
            rocketAnim.stop()
        }
    }
}
```

소스 코드 – 7쪽



Tween animation (1/2)

- **Tween animation**을 실행
  - Tween은 between의 시적 표현
  - 대상의 초기 상태와 마지막 상태를 지정하여 계산을 통해 중간 장면들을 연속적으로 생성하는 방식
    - 애니메이션 명령은 변환 종류, 변환이 발생하는 시간, 변환 지속 시간 등을 정의

- Tween animation은 주로 XML을 이용
  - <alpha>, <scale>, <translate>, <rotate>, <set>
  - 모든 애니메이션 명령은 동시에 적용됨
    - 순차적으로 적용하려면 startOffset 속성 설정

Tween animation (2/2)

- **위치 이동, 크기 변환, 회전 등의 효과가 적용된 중간 프레임을 생성**
  - Frame animation 에 비해 CPU는 더 많이 사용하지만
  - 메모리는 훨씬 적게 차지
- 구현 방법
  - XML에서 해당 속성을 지정하거나 AnimationSet 클래스를 사용

- **Alpha** (투명도 변환)
  - fromAlpha, toAlpha
- **Scale** (크기 변환)
  - fromXScale, toXScale, fromYScale, toYScale
  - pivotX, pivotY, fillAfter, fillBefore
- **Rotate** (회전)
  - fromDegrees, toDegress, toYScale
  - pivotX, pivotY, startOffset
- **Translate** (위치 이동)
  - toXDelta, toYDelta

실습 16: Tween animation 정의(1/2)

drawable /rect.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="20dp" />
    <stroke
        android:width="3dp"
        android:color="#ffffff"/>

</shape>
```

소스 코드 – 8쪽

anim/rotate.xml

res > 오른쪽 버튼 > New > Android Resource Directory Resource type → anim

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:interpolator="@android:anim/accelerate_interpolator">
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="6000" />
</set>
```

소스 코드 – 8쪽

anim/translate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator">
    <translate
        android:fromYDelta="0"
        android:toYDelta="100%"
        android:duration="3000"
    />
</set>
```

소스 코드 – 8쪽

실습 16: Tween animation 정의(2/2)

anim/scale.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator">
    <scale
        android:fromXScale="1.0"
        android:toXScale="0.1"
        android:fromYScale="1.0"
        android:toYScale="0.1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="2000"
    />
</set>
```

소스 코드 - 8쪽

anim/alpha.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator">
    <alpha
        android:fromAlpha="1.0"
        android:toAlpha="0.0"
        android:duration="1000"
        android:repeatCount="1"
        android:repeatMode="reverse"
    />
</set>
```

소스 코드 - 9쪽

잠깐! Tween Animation (1/2)

```
<rotate
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="6000"/>
```

fromDegrees, **toDegrees** : 회전 시작 각도와 끝 각도.
➔ 시계 방향으로 회전하는 360도 각도로 지정
pivotX, **pivotY** : 회전 중심
➔ 회전 중심을 생략하면 좌측 상단을 기준으로 회전.

```
<translate
    android:fromYDelta="0"
    android:toYDelta="100%"
    android:duration="3000"/>
```

출발 위치와 도착 위치를 값 또는 비율로 지정.
fromXDelta ➔ toXDelta
fromYDelta ➔ toYDelta

잠깐! Tween Animation (2/2)

```
<scale
    android:fromXScale="1.0"
    android:toXScale="0.1"
    android:fromYScale="1.0"
    android:toYScale="0.1"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="2000"/>
```

가로, 세로 방향 각각에 대해 **시작 배율과 끝 배율**을 지정.
확대 배율 : 실수로 표시. 1.0은 원본과 같은 크기.
1.0보다 크면 확대, 작으면 축소이다.
확대도 회전과 마찬가지로 **확대 중심(pivotX, pivotY)**을 지정할 수 있다.

android. startOffset = "3000"

Animation 시작 시간을 3초 후로 설정
일정 시간 경과 후에 animation을
보여주고 싶을 때 사용

```
<alpha
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:duration="1000"
    android:repeatCount="1"
    android:repeatMode="reverse"/>
```

시작 투명도와 끝 투명도를 지정
➔ 0~1사이의 값. 0은 완전 투명이며 1은 불투명.

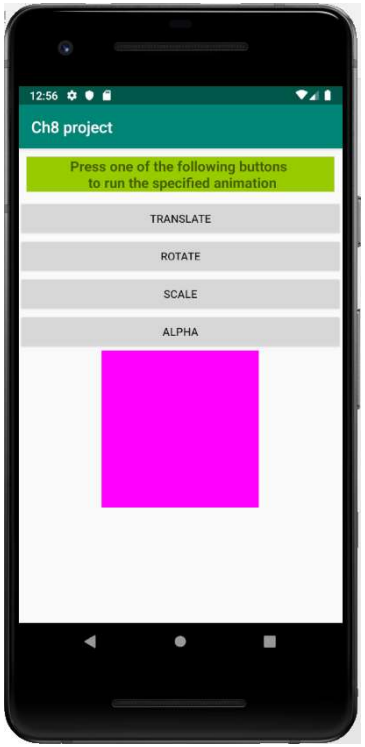
실습 16: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linear"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Press one of the followi
                to run the specific animation!
    <Button
        android:id="@+id/translate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="mOnClick"
        android:text="Translate Animation" />
    <Button
        android:id="@+id/rotate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="mOnClick"
        android:text="Rotate Animation" />
    <Button
        android:id="@+id/scale"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="mOnClick"
        android:text="Scale Animation" />
    <Button
        android:id="@+id/alpha"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="mOnClick"
        android:text="Alpha Animation" />

    <ImageView
        android:id="@+id/rect_image"
        android:layout_gravity="center_horizontal"
        android:layout_width="200dp"
        android:layout_height="100dp"
        android:background="@drawable/rect"/>
</LinearLayout>
```

소스 코드 – 9~10쪽

실습 16: Tween Animation

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        rect_image.setBackgroundColor(Color.MAGENTA)
    }

    fun onClick(v: View) {
        var anim: Animation? = null

        when (v.id) {
            R.id.translate -> anim =
                    AnimationUtils.loadAnimation(this, R.anim.translate)
            R.id.rotate -> anim =
                    AnimationUtils.loadAnimation(this, R.anim.rotate)
            R.id.scale -> anim =
                    AnimationUtils.loadAnimation(this, R.anim.scale)
            R.id.alpha -> anim =
                    AnimationUtils.loadAnimation(this, R.anim.alpha)
        }
        rect_image.startAnimation(anim)
    }
}
```



소스 코드 – 10쪽

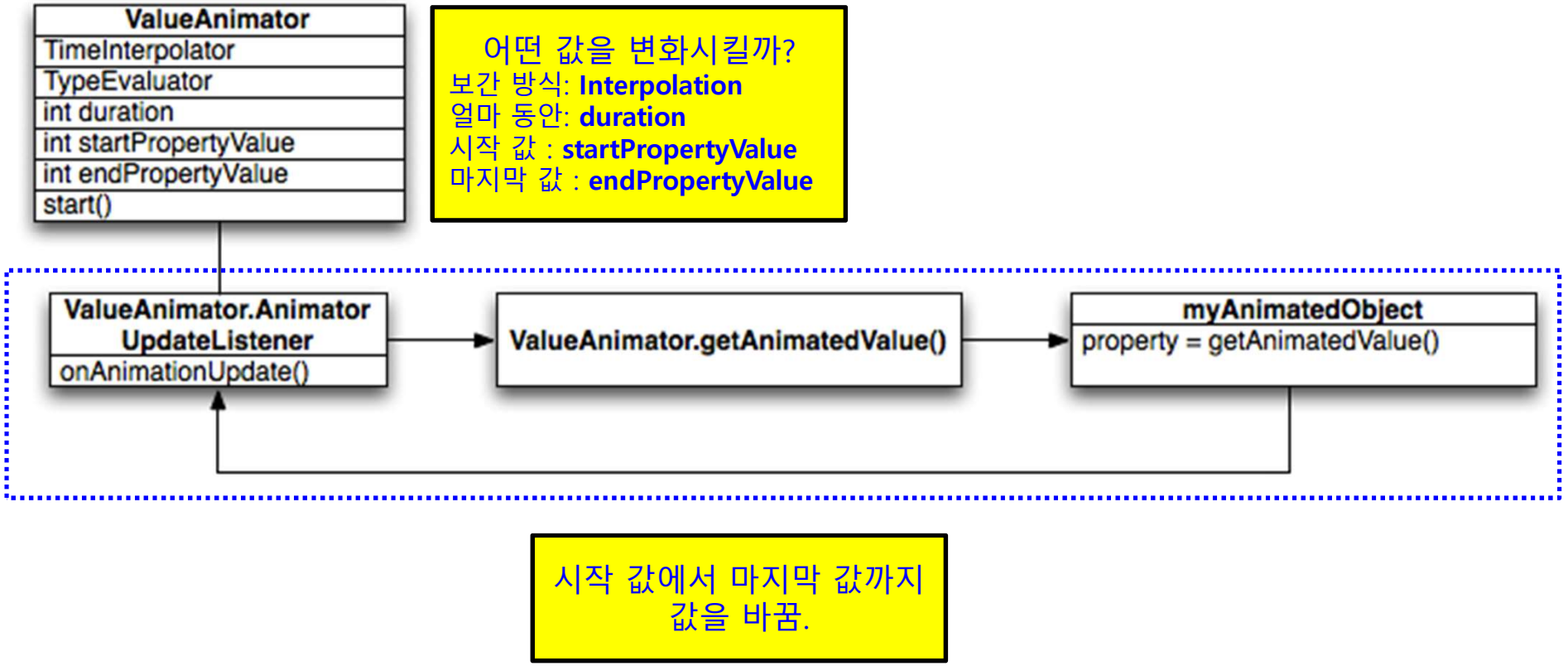
잠깐! Interpolator

- **AccelerateDecelerate**Interpolator
- **AnticipateOvershoot**Interpolator
- **Accelerate**Interpolator, **Decelerate**Interpolator
- **Anticipate**Interpolator
  - 역방향으로 움직였다가 다시 정해진 방향으로 이동
  - 새총과 유사한 효과
- **Bounce**Interpolator
- **Cycle**Interpolator
  - 지정된 회수만큼 animation을 반복.
- **Linear**Interpolator
  - 일정한 속도로 animation을 실행.
- **Overshoot**Interpolator
  - 지정된 목표지점을 지나쳤다가 복귀.

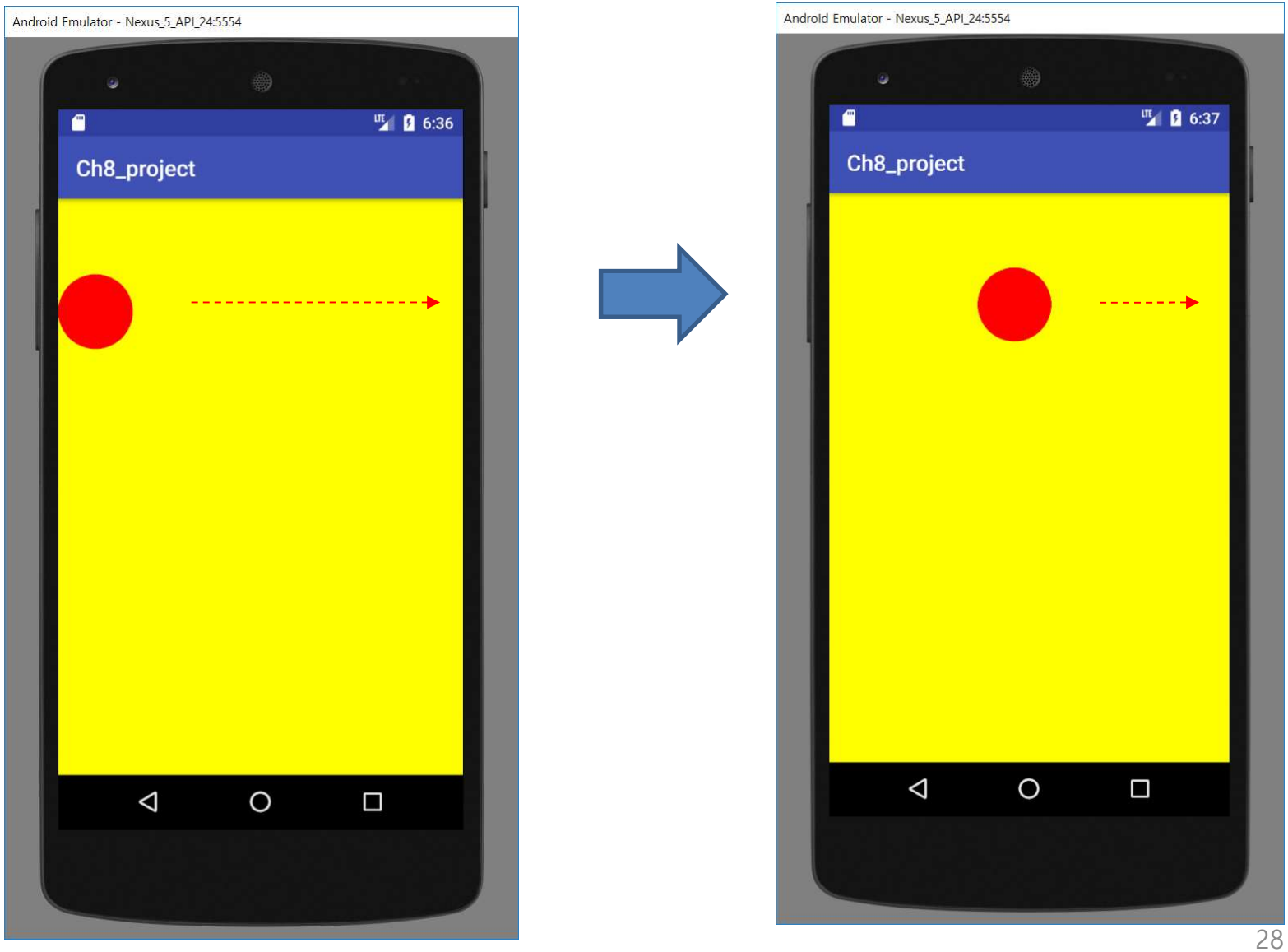
Property animation (1/2)

- 시간에 따라 객체의 속성을 변화시킴
  - 속성 값의 변화 범위 설정
    - 시작 값, 마지막 값
  - 시간 보간(time interpolation)
    - 시간 경과에 따라 값이 어떻게 변하는지 정의
  - 지속 시간(duration)

Property animation (2/2)

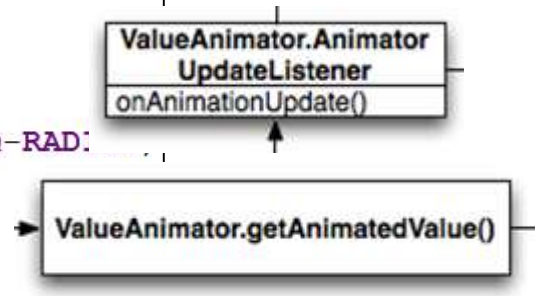


실습 17: x 축 방향으로 원 이동



실습 17: Property Animation

```
class MyView(context: Context): View(context) {
    val RADIUS:Float = 100f
    private var mX:Float = RADIUS

    override fun onDraw(canvas: Canvas?) {
        canvas?.drawARGB(255, 255, 255, 0)
        val paint = Paint()
        paint.color = Color.RED
        canvas?.drawCircle(mX, 300f, RADIUS, paint)
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        if (event.action == MotionEvent.ACTION_DOWN) {
            val valueAnim = ValueAnimator.ofFloat(RADIUS, width-RAD

            valueAnim.duration = 2000
            valueAnim.interpolator = AccelerateInterpolator()
            valueAnim.start()

            valueAnim.addUpdateListener{
                mX = it.animatedValue as Float
                invalidate()
            }
            return true
        }
        return false
    }
}
```




```
valueAnim.addUpdateListener(object: ValueAnimator.AnimatorUpdateListener {
    override fun onAnimationUpdate(animation: ValueAnimator?) {
        mX = animation?.animatedValue as Float
        invalidate()
    }
})
```

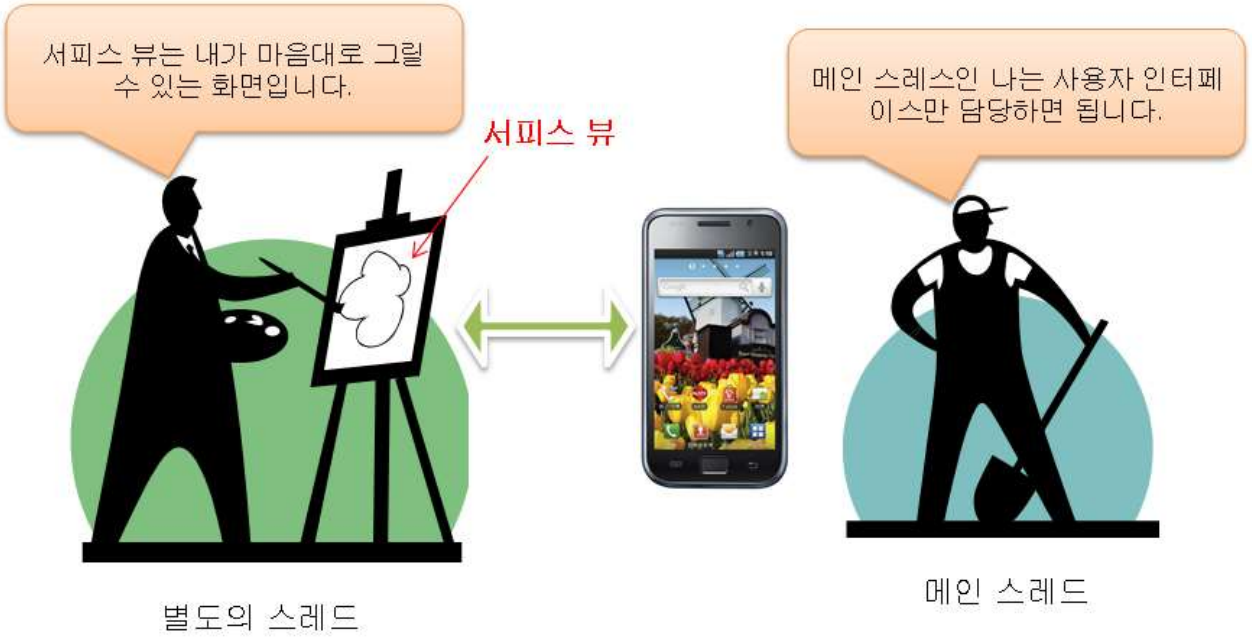
소스 코드 – 11~12쪽

What to do next?

- 도형 객체
- 애니메이션 기초
  - Property animation
  - View animation
  - Drawable animation
- **SurfaceView**

SurfaceView 개요(1/2)

- 사용자 인터페이스와는 달리
  - application에게 그림을 그릴 수 있는 별도 공간을 제공

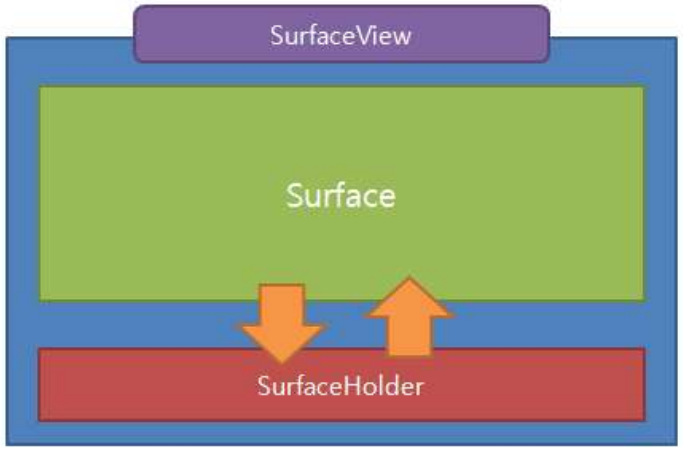


일반 View는 Canvas를 갖지만, SurfaceView는 surface(가상 화면,메모리)를 갖는다.

Surface는 실제 화면과 구조가 같기 때문에, 똑같은 방식으로 그리면 된다.

SurfaceView 개요(2/2)

- graphic 출력은 main thread가 직접 처리
  - main thread가 그래픽 작업량이 많은 게임에 CPU를 집중 사용하면, 사용자 입력 처리 속도가 느려진다.
    - Background thread에서 처리하도록 하면?
      - Thread는 화면에 출력할 수 없다!!!
  - Thread를 만들어 **가상 화면인 Surface**에 그래픽을 출력
    - SurfaceView가 surface에 그리고 있는 동안
      - Main Thread는 사용자 입력을 처리
    - Surface에 그림을 다 그렸으면,
      - Main thread에게 이를 알린다.



SurfaceView를 사용하려면?

- SurfaceView를 상속받는 View 클래스를 생성한다.
  - SurfaceView가 만들어졌다고 당장 그릴 수 없음!
    - SurfaceView가 준비되었을 때만 그릴 수 있으며,
    - SurfaceView가 destroy되었을 때는 그릴 수 없다.

- Main Thread가 SurfaceView에게 그려도 좋다고 어떻게 알려줄까?
  - **SurfaceHolder.Callback** 인터페이스의 callback 메소드 구현
    - **surfaceCreated, surfaceDestoryed, surfaceChanged**

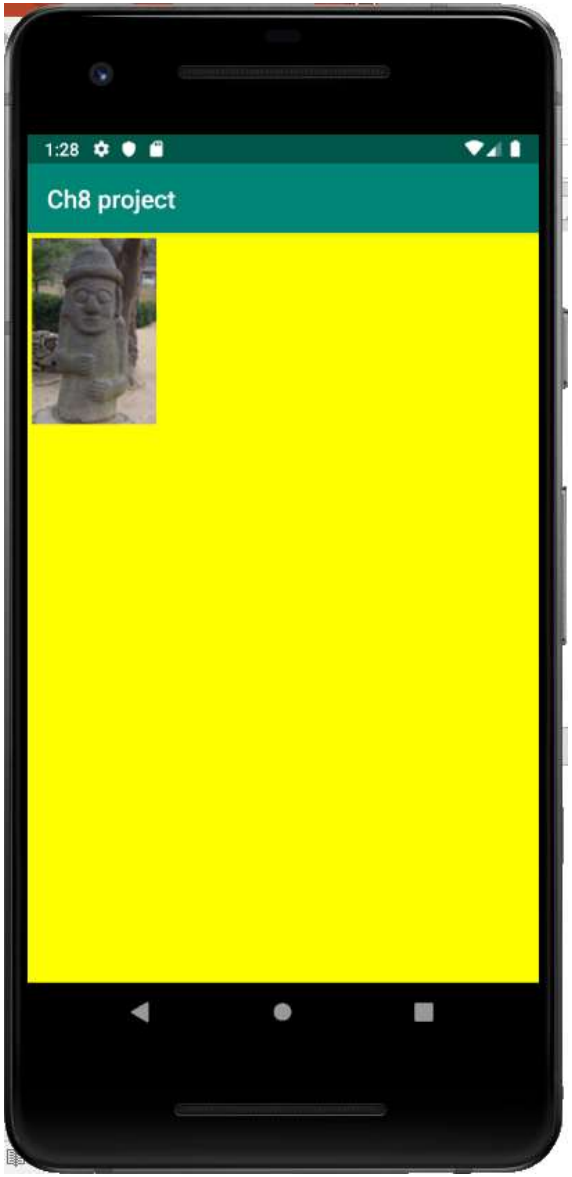
  - Surface 관리는 **SurfaceHolder** 객체가 담당
    - Surface에 변동이 생겼을 때 이를 처리할 callback 객체를 등록

    mHolder : SurfaceHolder = **holder**
    mHolder. addCallback (this)

SurfaceView 구조

```
class MySurfaceView (context :Context) : SurfaceView (context),
                        SurfaceHolder. Callback {
    override fun surfaceCreated (holder : SurfaceHolder) {
            // surface가 준비되었으므로 surface에 그릴 수 있음
    }
    override fun surfaceDestroyed (holder : SurfaceHolder) {
            // surface가 소멸되었으므로 그리기 즉시 중단
    }
    override fun surfaceChanged (holder : SurfaceHolder,
                        format : Int, width : Int, height : Int) {
            // surface가 바뀌었기 때문에 표면 크기 초기화
    }
}
```

실습 18: 단계별 SurfaceView 예제 구현



Step 1: 이미지 출력

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(MySurfaceView(this))
    }
}
```

MySurfaceView.kt

```
class MySurfaceView(context: Context): View(context) {
    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)

        val b: Bitmap = BitmapFactory.decodeResource(
            resources, R.drawable.harubang
        )
        if (canvas != null) {
            canvas.drawColor(Color.YELLOW)
            canvas.drawBitmap(b, 10f, 10f, null)
        }
    }
}
```

Step 2: SurfaceView 클래스 상속

생성자에 SurfaceHolder 객체를 생성하고,
surface 상태 정보를 전달받기 위해 callback 객체를 등록.

```
class MySurfaceView(context: Context): SurfaceView(context) {
    private val mHolder:SurfaceHolder = holder

    init {
        mHolder.addCallback(this)
    }

    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)

        val b: Bitmap = BitmapFactory.decodeResource(
            resources, R.drawable.harubang
        )
        if (canvas != null) {
            canvas.drawColor(Color.YELLOW)
            canvas.drawBitmap(b, 10f, 10f, null)
        }
    }
}
```

에러 발생! Why?
SurfaceHolder.Callback 인터페이스를 구현한 객체이어야 함.

Step 3: callback 인터페이스 상속

```
class MySurfaceView(context: Context): SurfaceView(context),
                            SurfaceHolder.Callback {
    private val mHolder:SurfaceHolder = holder

    init {
        mHolder.addCallback(this)
    }

    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)

        val b: Bitmap = BitmapFactory.decodeResource(
            resources, R.drawable.harubang
        )
        if (canvas != null) {
            canvas.drawColor(Color.YELLOW)
            canvas.drawBitmap(b, 10f, 10f, null)
        }
    }

    override fun surfaceChanged(holder: SurfaceHolder?,
                                format: Int, width: Int, height: Int) {}

    override fun surfaceDestroyed(holder: SurfaceHolder?) {}

    override fun surfaceCreated(holder: SurfaceHolder?) {}
}
```

3개의 callback 메소드를 구현해야 함!

Step 4: surfaceCreated 메소드

```
override fun surfaceCreated(holder: SurfaceHolder?) {
    var c: Canvas? = null
    try {
        c = mHolder.lockCanvas()
        synchronized(mHolder) {
            val b: Bitmap = BitmapFactory.decodeResource(
                resources, R.drawable.harubang)
            c.drawColor(Color.YELLOW)
            c.drawBitmap(b, 10f, 10f, null)
        }
    } finally {
        if (c != null) {
            mHolder.unlockCanvasAndPost(c)
        }
    }
}
```

원래 onDraw 메소드 내용과 Synchronized 블록의 코드 내용과 비교해 볼 것!

```
override fun onDraw(canvas: Canvas?) {
    super.onDraw(canvas)

    val b: Bitmap = BitmapFactory.decodeResource(
        resources, R.drawable.harubang
    )
    if (canvas != null) {
        canvas.drawColor(Color.YELLOW)
        canvas.drawBitmap(b, 10f, 10f, null)
    }
}
```

완성된 SurfaceView 구현

```
class MySurfaceView(context: Context) : SurfaceView(context),
                                SurfaceHolder.Callback {
    private val mHolder:SurfaceHolder = holder

    init {
        mHolder.addCallback(this)
    }

    override fun surfaceChanged(holder: SurfaceHolder?,
                                format: Int, width: Int, height: Int) {
    }

    override fun surfaceDestroyed(holder: SurfaceHolder?) {
    }

    override fun surfaceCreated(holder: SurfaceHolder?) {
        var c: Canvas? = null
        try {
            c = mHolder.lockCanvas(null)
            synchronized(mHolder) {
                val b = BitmapFactory.decodeResource(
                    resources, R.drawable.harubang)
                c.drawColor(Color.YELLOW)
                c.drawBitmap(b, 10f,10f, null)
            }
        } finally {
            if (c != null) {
                mHolder.unlockCanvasAndPost(c)
            }
        }
    }
}
```

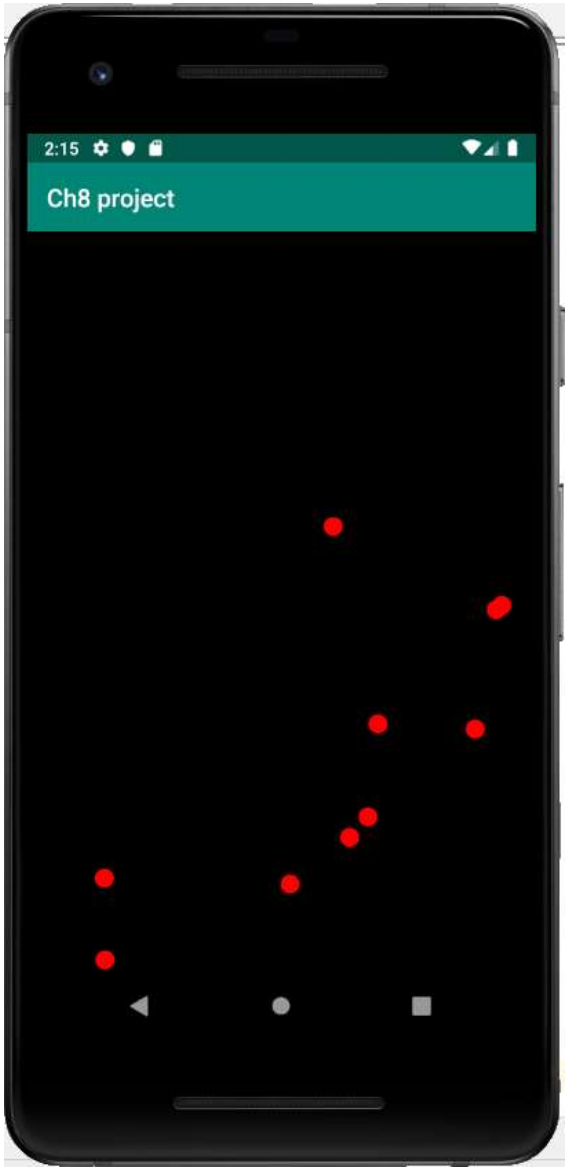
onDraw 메소드는 반드시 순삭!!

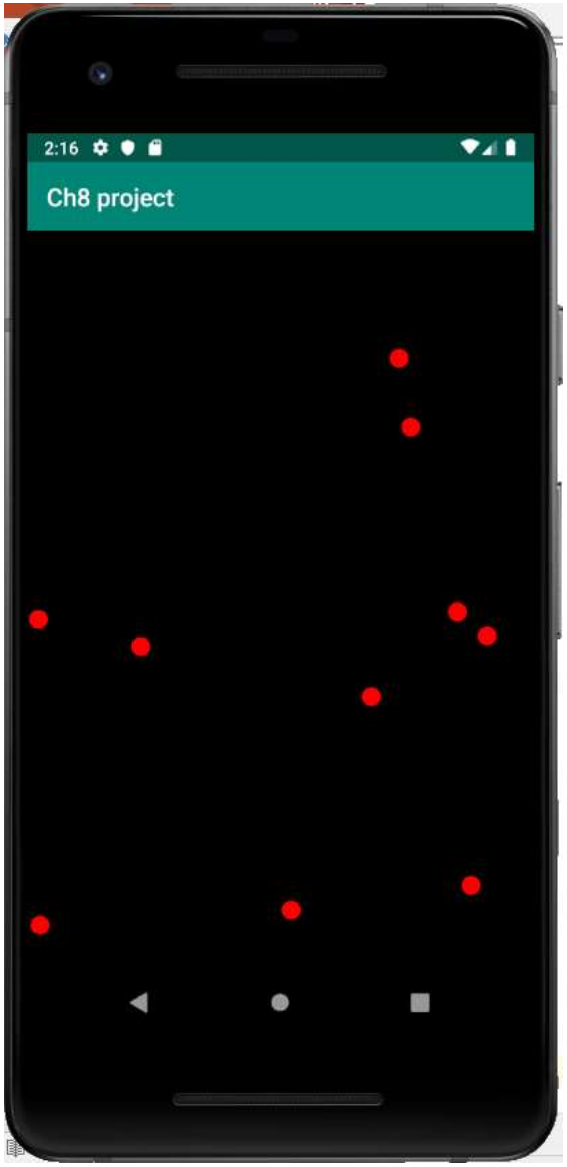
소스 코드 – 13쪽

잠깐 !

```
try {
    // surface 사용 권한을 얻는다.
    // 권한을 얻으면, Canvas 객체 c를 가져올 수 있다.
    c = mHolder.lockCanvas()
    synchronized(mHolder) {
        // surface 에 그림을 그린다.
        val b: Bitmap = BitmapFactory.decodeResource(
            resources, R.drawable.harubang)
        c.drawColor(Color.YELLOW)
        c.drawBitmap(b, 10f, 10f, null)
    }
} finally {
    if (c != null) {
        // surface 에 그린 그림을 장치에 복사한다.
        // surface 사용 권한을 해지한다.
        mHolder.unlockCanvasAndPost(c)
    }
}
```

실습 19: Bounce Balls





실습 19: 3 kt classes

MainActivity. kt

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(MySurfaceView(this))
    }
}
```

Ball. kt

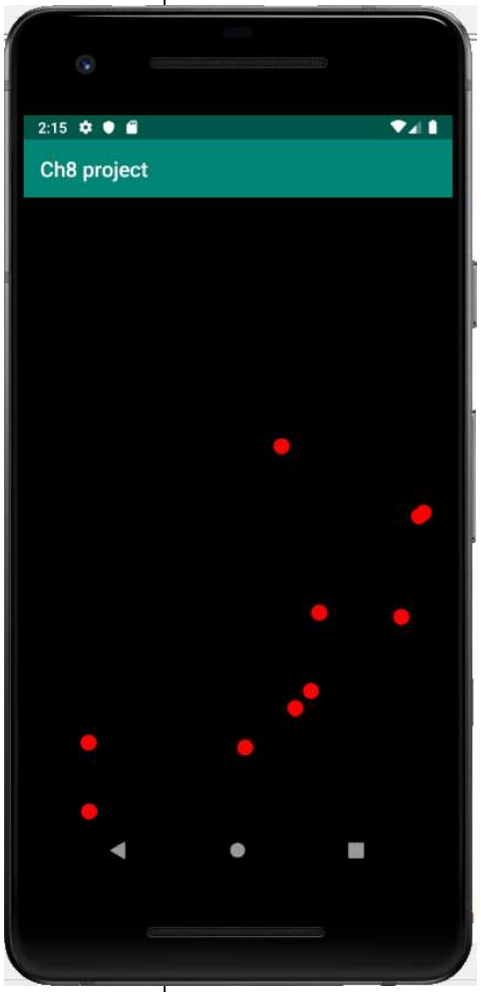
```
class Ball(var radius:Float, var width:Float, var height:Float) {
    var x:Float = 0f
    var y:Float = 0f
    var xInc = 1f
    var yInc = 1f

    init {...}

    fun drawBall(c: Canvas) {...}
```

MySurfaceView. kt

```
class MySurfaceView  ... {
  var basket = arrayOfNulls <Ball>[10]
  private val thread MyThread

  override fun surfaceCreated ( . . . )
  override fun surfaceChanged (  . . . )
  override fun surfaceDestroyed ( . . . )
}
```

실습 19: Ball.kt

```
class Ball(var radius:Float, var width:Float, var height:Float) {
    var x:Float = 0f
    var y:Float = 0f
    var xInc = 1f
    var yInc = 1f

    init {
        x = genRandomNumber(width - radius)
        y = genRandomNumber(height - radius)
        xInc = genRandomNumber(10f)
        yInc = genRandomNumber(10f)
    }

    fun drawBall(c: Canvas) {
        val paint = Paint()

        if (x < 0 || x > width - radius) xInc = -xInc
        if (y < 0 || y > height - radius) yInc = -yInc
        x += xInc
        y += yInc
        paint.color = Color.RED
        c.drawCircle(x, y, radius, paint)
    }

    private fun genRandomNumber(range:Float):Float {
        return (Math.random() * range).toFloat()
    }
}
```



소스 코드 – 14쪽

실습 19: MySurfaceView.kt - Overview

```
class MySurfaceView(context: Context) : SurfaceView(context),
                        SurfaceHolder.Callback {
    private val mHolder: SurfaceHolder = holder
    var thread: MyThread
    var basket = arrayOfNulls<Ball>(10)

    init {...}

    inner class MyThread : Thread() {...}

    override fun surfaceCreated(holder: SurfaceHolder) {...}

    override fun surfaceChanged(holder: SurfaceHolder,
                                format: Int, width: Int, height: Int) {...}

    override fun surfaceDestroyed(holder: SurfaceHolder) {...}
```

소스 코드 – 14~16쪽

실습 19: MySurfaceView.kt (1/3)

```
class MySurfaceView(context: Context) : SurfaceView(context),
                        SurfaceHolder.Callback {
    private val mHolder: SurfaceHolder = holder
    var thread: MyThread
    var basket = arrayOfNulls<Ball>(10)

    init {
        mHolder.addCallback(this)

        thread = MyThread()

        val dm = resources.displayMetrics
        val w = dm.widthPixels.toFloat()
        val h = dm.heightPixels.toFloat() - getActionBarSize()
        for (i in 0..9) {
            basket[i] = Ball(20f,w, h)
        }
    }
```

```
private fun getActionBarSize():Float {
    val styledAttributes =
        context.theme.obtainStyledAttributes(
            intArrayOf(R.attr.actionBarSize))
    val size:Float = styledAttributes.getDimension(0, 0f)
    styledAttributes.recycle()
    return size
}
```

실습 19: MySurfaceView.kt (2/3)

```
override fun surfaceCreated(holder: SurfaceHolder) {
    thread.start()
}

override fun surfaceChanged(holder: SurfaceHolder,
                            format: Int, width: Int, height: Int) {
}

override fun surfaceDestroyed(holder: SurfaceHolder) {
    while (true) {
        try {
            thread.join()
            break
        } catch (e: InterruptedException) { }
    }
}
```

여러 개 스레드가 만들어져
(fork) 실행되다가
스레드 실행을 중지할 때
호출하는 메소드가 join

실습 19: MyThread – 내부 클래스 (3/3)

```
inner class MyThread : Thread() {
    override fun run() {
        while (true) {
            var c: Canvas? = null
            try {
                c = mHolder.lockCanvas(null)
                c.drawColor(Color.BLACK)
                synchronized(mHolder) {
                    for (b in basket) {
                        b?.drawBall(c)
                    }
                }
            } finally {
                if (c != null) {
                    mHolder.unlockCanvasAndPost(c)
                }
            }
        }
    }
}
```