

```

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

    // (1)
    var myTextView : TextView = findViewById(R.id.my_textview)

    // (2)
    val greetingMsg = resources.getText(R.string.greeting).toString()
}

```


(1) 뷰 객체를 반환받기 위해 사용하는 `findViewById` 메서드를 호출하며, `TextView`의 식별자 값을 전달하고 해당 뷰 객체를 반환받습니다.

(2) `greeting` 문자열 리소스에 접근합니다. 액티비티에 포함된 `resources` 속성에 접근해 `getText` 메서드를 호출하고 문자열 리소스의 식별자 값을 전달해서 해당 문자열 값을 반환받습니다.

다음은 색상 리소스 파일(`colors.xml`)에 추가한 색상 리소스에 접근하는 예입니다.

```
val redColor = ContextCompat.getColor(applicationContext, R.color.red)
```

`ContextCompat` 클래스에서 제공하는 `getColor` 메서드를 호출하며, 색상 리소스의 식별자 값을 전달해서 색상과 관련된 정숫값을 가져옵니다. 리소스의 타입에 따라 접근하는 데 사용하는 클래스나 메서드는 조금씩 달라도 공통적으로 리소스에 부여된 식별자 값을 이용해 리소스에 접근하는 것을 확인할 수 있습니다.

 서로 다른 레이아웃 파일에서는 뷰에 같은 식별자를 써도 문제가 발생하지 않습니다. 하지만 다른 레이아웃 파일에 정의한 뷰에도 가급적 겹치지 않는 완전히 고유한 식별자를 부여하는 것을 권장합니다.

안드로이드에서 사용되는 가상 단위 DP

일반적으로 디지털 화면에 무언가를 표시할 때 크기를 지정하는 단위로 자주 사용하는 단위는 `px`(픽셀)입니다. 픽셀은 화면을 구성하는 최소 단위로 화면에 표시될 하나의 화소를 기준으로 삼는 단위입니다. 안드로이드에서도 `px` 단위로 뷰 또는 뷰그룹의 크기를 지정할 수 있지만 앞으로 설명할 문제점 때문에 `px` 단위는 거의 사용되지 않습니다.

px 단위를 사용하면 생기는 문제점을 설명하기 위해 가로, 세로 10px 크기의 정사각형 화면에 5px 크기의 사각형을 그리는 상황을 가정해보겠습니다. 화면에 그려진 사각형은 다음과 같이 표시될 것입니다.

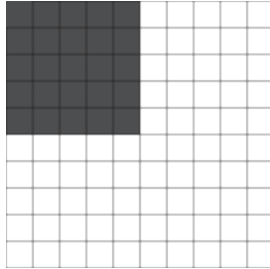


그림 1-46 가로, 세로 10px 크기의 사각형에 그려진 5px 사각형

화면의 가로, 세로 크기가 20px로 두 배가 됐다고 가정하고 똑같은 5px 크기의 사각형을 그린다면 사각형은 다음과 같이 표시될 것입니다.

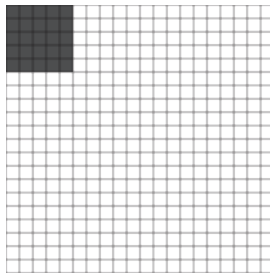


그림 1-47 두 배 크기의 사각형에 그려진 5px 사각형

문제는 해상도(화면의 크기)가 높아질 때 발생합니다. 해상도가 높은 화면에 같은 px 단위의 크기를 가진 대상을 그렸을 때 화면에 보이는 상대적 크기가 더 작아 보이는 현상이 발생합니다. 만약 파란색 사각형이 버튼이었다면 높아진 해상도로 인해 상대적으로 크기가 줄어드는 바람에 버튼을 클릭하기가 무척 불편해지는 상황이 발생할 것입니다.

화면의 해상도를 높여서 더 많은 화소를 표시할 수 있다면 상대적으로 더 선명한 화질을 보장할 수 있으므로 단말기 제조사에서는 계속해서 단말기의 해상도를 높일 수 있는 방안을 연구하고 있습니다. 이러한 상황에서 px 단위를 사용해 화면에 대상을 표시할 경우 상대적으로 비슷한 크기로 표시해야 할 대상이 고해상도 단말기에서는 매우 작게 보이는 문제를 야기할 수 있습니다.

다음은 최초로 출시된 안드로이드 단말기인 HTC Dream 및 갤럭시 S1과 비교적 최근에 출시된 S10의 해상도를 정리한 표입니다. HTC Dream의 가로 해상도(320)와 갤럭시 S10의 가로 해상도(1440)를 비교하면 크기가 거의 4배 넘게 차이 나는 것을 확인할 수 있습니다.

모델	해상도
HTC Dream (최초의 안드로이드 폰)	320 × 480
갤럭시S1	480 × 800
갤럭시S10	1440 × 3040

앞에서 살펴본 문제를 해결하기 위해 안드로이드에서는 dp(density-independent pixel)라고 하는 **가상의 단위를** 제공합니다. 이름에서 유추할 수 있듯이 **화면의 해상도에 영향을 받지 않는 픽셀**이라는 의미를 가지고 있습니다.

다시 가로, 세로의 크기가 10px인 정사각형 화면에 5dp 크기의 사각형을 그리는 상황을 가정하면 사각형은 다음과 같이 표시될 것입니다.

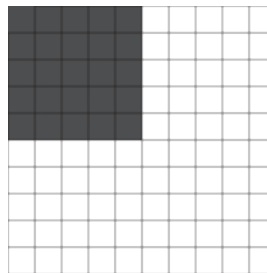


그림 1-48 가로, 세로 10px 크기의 사각형에 그려진 5dp 사각형

화면 크기가 두 배가 됐다고 가정하고 똑같은 **5dp 크기의 사각형**을 그리면 사각형은 다음과 같이 표시될 것입니다.

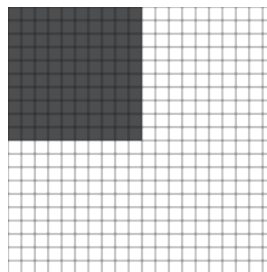


그림 1-49 두 배 크기의 사각형에 그려진 5dp 사각형

상대적인 크기가 같아 보일 수 있도록 실제로는 10px 크기의 사각형이 그려진 것을 확인할 수 있습니다. 화면의 해상도가 두 배가 됐으므로 px 단위로 그려지는 사각형의 크기에 두 배의 **확대 비율**을 적용해 사각형을 그리면 화면에서의 상대적인 크기에서 차이가 생기지 않는다는 것을 알 수 있습니다.

정리하면 화면의 해상도가 높아질 경우 화면의 해상도(화면 밀도)에 비례해 특정 **확대 비율**을 곱해서 대상을 그리면 화면에서의 상대적 크기에는 차이가 나지 않을 것이라는 결론을 내릴 수 있습니다. 단, 화면의 해상도에 따라 어느 정도의 확대 비율을 곱해야 하는지에 대해서는 기준이 필요합니다.

화면의 해상도에 따라 어느 정도의 확대 비율을 적용해야 하는가에 대한 기준을 부여하기 위해 안드로이드에서는 다음과 같이 **화면 밀도에 따른 구분 단위(density bucket)**를 제공합니다.

화면 밀도에 따른 구분 단위	화면 밀도	확대 비율(mdpi 대비)
mdpi	~160dpi	1.0
hdpi	~240dpi	1.5
xhdpi	~360dpi	2.0
xxhdpi	~480dpi	3.0
xxxhdpi	~640dpi	4.0

여기서 등장하는 dpi 단위는 dots per inch의 약어로, 1인치(2.54cm) 크기의 사각형에서 가로, 세로로 몇 개의 픽셀(화소)이 존재하는지를 표시하는 단위입니다.

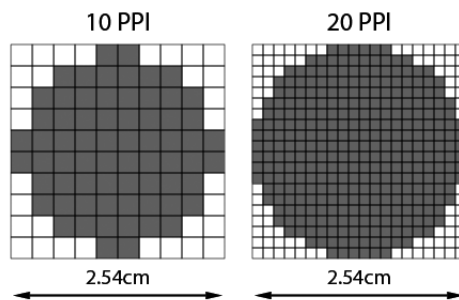


그림 1-50 서로 다른 DPI를 가진 화면에 그려진 원

가령 화면 밀도가 mdpi인 단말기에서는 물리적인 화면의 2.54cm 크기의 사각형 내부에 최대 25600(160x160)개의 픽셀이 표시됩니다. dpi가 높다면 물리적으로 같은 크기의 화면에 더 많은 화소를 표시할 수 있기 때문에 더 선명한 화면을 제공할 수 있습니다.

여기서 눈여겨보아야 할 구분 단위는 mdpi입니다. 이 구분 단위에서는 1px이 1dp의 크기로 변환되므로 따로 확대 비율을 적용하지 않아도 됩니다. 이러한 특성 때문에 mdpi는 기준이 되는 구분 단위(baseline density)로 사용됩니다. 만약 화면 밀도가 세 배인 xxhdpi 구분 단위의 단말기에서 mdpi에서 표시하던 이미지를 표시해야 한다면 세 배의 확대 비율을 적용해야 합니다.

하지만 dp 단위를 사용해도 이는 어디까지나 가상의 단위이기 때문에 실제 화면에는 모든 대상이 픽셀 단위로 표시됩니다. 따라서 다음과 같이 단말기의 화면 밀도에 따라 확대 비율이 적용된 여러 별의 이미지를 준비해야 합니다.



그림 1-51 화면 밀도에 따라 크기가 다르게 적용된 아이콘 이미지들

가령 mdpi 해상도에서 32dp 크기로 표시하기 위해 준비한 32px 크기의 아이콘 이미지를 xxhdpi 해상도에서 똑같은 32dp 크기로 표시하려면 해당 이미지를 세 배 확대한 96px 크기의 이미지를 제공해서 표시해야 단말기 해상도에 걸맞은 화질로 보여질 것입니다.

단말기의 해상도에 따르는 여러 별의 이미지를 준비해야 하므로 보통 큰 사이즈의 이미지 리소스를 먼저 만든 후 해당 이미지를 축소해서 더 낮은 화면 밀도에 사용할 이미지 리소스를 만듭니다.

이 책을 집필하고 있는 현재 시점에는 대부분의 최신 단말기가 xxhdpi 해상도를 지원합니다.

앞에서 리소스 파일을 저장할 폴더의 특징을 살펴보면서 **drawable** 폴더에 화면에 표시할 이미지 파일을 저장한다고 설명한 바 있습니다. 다양한 화면 밀도를 지원하기 위한 여러 해상도의 이미지를 제공하려면 다음과 같이 **drawable** 폴더명 뒤에 하이픈(-)과 구분 단위를 지정한 폴더를 생성하고 해당 폴더에 대응하는 확대 비율을 적용한 이미지를 저장하면 됩니다.



그림 1-52 화면 밀도에 따라 크기가 조정된 이미지를 저장할 폴더

이후 안드로이드 앱이 설치되고 실행되는 과정에서 단말기의 화면 밀도에 대응하는 리소스가 자동으로 선택되어 적용됩니다.

폴더 이름	저장 이미지
drawable-mdpi	확대 비율을 적용하지 않은 이미지를 저장
drawable-hdpi	1.5배의 확대 비율을 적용한 이미지를 저장
drawable-xhdpi	2배의 확대 비율을 적용한 이미지를 저장
drawable-xxhdpi	3배의 확대 비율을 적용한 이미지를 저장
drawable-xxxhdpi	4배의 확대 비율을 적용한 이미지를 저장

가령 drawable-mdpi 폴더에는 확대 비율이 적용되지 않은 이미지를, drawable-xxhdpi 폴더에는 세 배의 크기로 확대된 고해상도 이미지를 저장해두면 앱이 실행될 단말기의 화면 밀도에 대응하는 폴더에 포함된 이미지가 자동으로 적용됩니다.

Q 특정 화면 밀도를 지원하기 위한 drawable 폴더(예: drawable-xxhdpi)에만 리소스 이미지가 존재하는 경우 기계적인 리사이즈 과정을 거쳐 자동으로 상위, 하위 화면 밀도의 단말기에 필요한 이미지를 적용하게 됩니다. 하지만 가능하다면 화면 밀도에 따르는 여벌의 리소스를 모두 직접 리사이즈하는 방식으로 제작하는 것을 권장합니다.

이처럼 화면 밀도에 따른 여러 벌의 이미지를 준비하는 과정이 번거롭다면 해상도에 구애받지 않고 적용할 수 있는 벡터 형식의 이미지를 활용할 수도 있습니다. 벡터 이미지의 활용법은 이후 프로젝트를 진행하면서 살펴보겠습니다.

주로 글자 크기를 정할 때 사용하는 sp 단위를 설명하는 것으로 자주 쓰이는 단위에 대한 설명을 마무리하겠습니다.

sp는 scale-independent pixel의 약어로서 기본적으로는 dp 단위와 크게 다르지 않지만 단말기의 기본 글자 크기 설정에 영향을 받는다는 차이가 있습니다. 다음은 위에서부터 각각 16dp, 16sp 단위로 글자의 크기를 설정한 두 개의 TextView를 표시한 모습입니다.

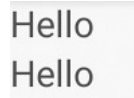


그림 1-53 단말기 글자 크기 조절 전 화면

글자 크기에 아무런 차이가 없고 똑같은 크기로 표시되는 것을 확인할 수 있습니다. 글자 크기에 차이를 주려면 단말기의 디스플레이 설정에서 기본 글자 크기를 더 크게 표시하도록 조정해야 합니다.

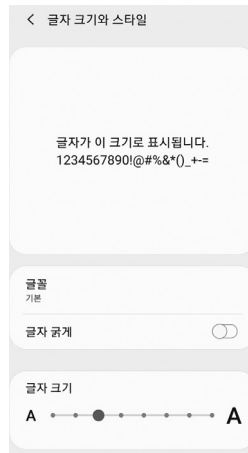


그림 1-54 단말기의 글자 크기 조절 화면

이후 다시 화면을 살펴보면 다음과 같이 단말기의 기본 글자 크기 설정에 영향을 받아 sp 단위를 적용한 TextView의 글자 크기가 상대적으로 더 커진 것을 확인할 수 있습니다.

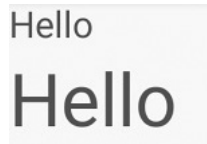


그림 1-55 단말기 글자 크기 조절 후 화면

일반적으로 글자 크기를 지정할 때는 단말기의 기본 글자 크기 설정을 존중해서 적절히 글자 크기가 조정될 수 있도록 sp 단위를 사용하는 것이 좋습니다. 그러나 만약 글자 크기가 기본 글자 크기 설정과 상관없이 균일한 크기로 보여지길 원한다면 sp 단위가 아닌 dp 단위를 사용해야 합니다.

첫 액티비티 코드 분석과 수정

이제 첫 번째로 생성한 프로젝트의 구조를 살펴본 후 코드를 수정해 안드로이드 앱의 기초적인 동작 방식을 본격적으로 살펴보겠습니다.

안드로이드의 소스 코드는 **java** 폴더 안의 **기본 패키지**에 저장됩니다. 프로젝트를 생성하면 해당 패키지 내부에 **MainActivity** 파일이 자동으로 생성된 것을 확인할 수 있습니다. 이 액티비티는 앱이 실행될 때 첫 화면을 보여줄 **시작 액티비티**로 등록됩니다.

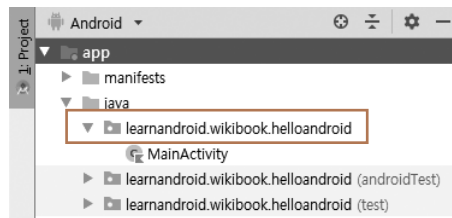


그림 1-56 코틀린 소스 저장 위치

자동 생성된 MainActivity 클래스의 내용은 다음과 같습니다.

/MainActivity.kt

```
// (1)
class MainActivity : AppCompatActivity() {
    // (3)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // (2)
        setContentView(R.layout.activity_main)
    }
}
```

(1) 액티비티(AppCompatActivity) 클래스를 상속받는 MainActivity 클래스가 정의돼 있습니다. 액티비티는 앱을 구성하는 기본 요소 중 하나로 화면에 보여줄 하나의 화면을 구성하는 용도로 사용됩니다.