

Hanbit eBook

Realtime 68



개발 프로세스 향상과 코드의 질을 높이는

Android Developer Tools 필수 가이드

심화편

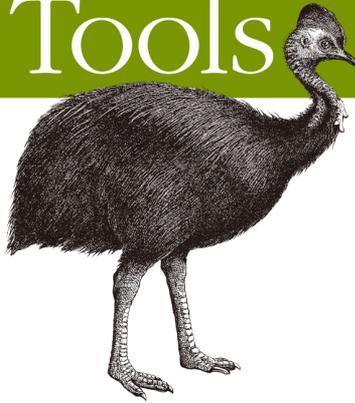
Android Developer Tools

마이크 울프선 지음 / 이성주 옮김

O'REILLY®  한빛미디어
Hanbit Media, Inc.

Android Studio to Zipalign

Android Developer Tools



Essentials

O'REILLY®

Mike Wolfson

이 도서는 O'REILLY의
Android Developer Tools의
번역서입니다.

개발 프로세스 향상과 코드의 질을 높이는 **Android Developer Tools 필수 가이드 심화편**

초판발행 2014년 06월 03일

지은이 마이클 울프슨 / **옮긴이** 이성주 / **펴낸이** 김태헌

펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부

전화 02-325-5544 / **팩스** 02-336-7124

등록 1999년 6월 24일 제10-1779호

ISBN 978-89-6848-714-9 15000 / **정가** 11,000원

책임편집 배용석 / **기획·편집** 김창수

디자인 표지 여동일, 내지 스튜디오 [팀], 조판 최승실

영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 서은옥, 김옥현

이 책에 대한 의견이나 오타자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주세요.

한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea Copyright © 2013 HANBIT Media, Inc.

Authorized Korean translation of the English edition of Android Developer Tools Essentials, ISBN 9781449328214 © 2013 Mike Wolfson. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

이 책의 저작권은 오라일리사와 한빛미디어(주)에 있습니다.

저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.

책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **마이크 울프슨** Mike Wolfson

마이크 울프슨은 애리조나 피닉스 출신의 열정적인 모바일 디자이너/개발자다. 소프트웨어 업계에 20년째 몸담아 오고 있으며, 안드로이드를 초창기부터 다뤘다. 현재 건강 관리 분야의 안드로이드 앱을 개발하고 있다. 여러 성공적인 앱을 만들어 왔는데, 특히 'Droid Of the Day'가 가장 성공적인 앱이다.

마이크는 다른 사람들에게 기술을 가르치는 데 많은 시간을 할애해 왔다. 현재는 지역의 구글 개발자 그룹을 이끌고 있으며, 배움과 관련한 여러 다른 그룹에 참여하고 있다. 학회와 사용자 그룹에서 안드로이드와 모바일 개발 관련 강연을 하고 있다.

앱 개발을 하지 않을 때는 스노우보드, 하이킹, 스쿠버 다이빙을 하거나, PEZ Dispenser를 수집하거나, 어린(하지만 재빠른) 딸을 잡으러 다닌다.

역자 소개

옮긴이_ 이성주

인생의 대부분을 서울에서 보냈고, 그중 일부는 미국 캘리포니아, 텍사스, 뉴욕 등지에서 보냈다. 대학원 연구실 동료가 차린 벤처에서, 안드로이드(당시 2.2 버전)에서는 아이폰과 같은 동작이 힘들다는 것을 고객에게 설명하는 데 많은 시간을 할애했다(지금의 안드로이드 OS는 매우 좋다). 안드로이드 앱은 한국에 스마트폰이 들어왔을 때부터 다뤄왔다. 하지만, 삼성 갤럭시 2가 나온 당시에도 키보드 없는 스마트폰을 어떻게 쓰냐며 블랙베리를 사용했는데, 블랙베리 카카오톡 버전이 더 나아질 희망이 없다는 것을 오랫동안 인정하지 못했다.

지금은 대학과 기업에서 컴퓨터 과학, 소프트웨어 공학, 안드로이드를 비롯한 모바일 관련 주제로 강의를 주로 한다. 이 책처럼 좋은 책의 번역이 들어오면 번역도 하고 있다(물론 대박이 날 거라며 혼자서 생각하는 앱을 언제나 개발 중이다. 그 날이 오면 ...).

미국 저자가 쓴 기술서의 저자 소개에는 앱 개발을 안 할 때는 야외 활동을 활발하게 한다는 내용이 꼭 들어있다는 것이 늘 의아하다. 나도 안 하는 건 아니지만, 남들 하는 정도(중학교 때는 학교 대표 축구선수였다).

저자 서문

이 책을 선택했다면 안드로이드 개발에 대해 어느 정도 경험이 있을 것이고, 또 앱 개발이 꽤 까다롭다는 것을 이미 느끼고 있을 것이다. ‘안드로이드 개발자 도구(ADT, Android Developer Tools)’를 효과적으로 사용하면 개발 과정이 더욱 수월해질 수 있고, 코드의 품질도 높일 수 있어 보다 정제되고 안정적인 최종 결과물 제작이 가능하다.

안드로이드 개발자 도구(ADT) 필요 사항

안드로이드는 현존하는 다른 모바일 플랫폼들과 상당히 차별화된다. 한 조직에서 관장하는 것이 아니라 여러 회사가 참여하는 ‘Open Handset Alliance(OHA)’라는 그룹에서 관리한다. OHA는 무료 공개 소스이면서 완성된 형태의 모바일 운영 체제로 안드로이드를 제공한다. 이런 방식은 안드로이드 플랫폼이 특정 조직에 귀속되지 않도록 방지하지만, 이로 인해 다음과 같은 사항들은 좀 복잡해졌다.

다양한 화면 크기

안드로이드 기기들은 화면 크기가 다양하다. 다양한 크기의 화면에서 앱이 어떻게 표시되는지에 앱의 성공이 달려 있다.

파편화

통신사와 제조사가 새 운영체제와 실행환경(runtime) 버전 업데이트를 결정한다. 그래서 버전 업데이트가 제때 이루어지지 않는 경우도 많기 때문에 이전 버전의 운영체제 및 실행환경과 호환되도록 앱을 제작하는 것이 중요하다.

하드웨어별 사양 차이

안드로이드 기기들은 다양한 모양과 크기, 그리고 사양으로 출시된다. 하드웨어 사양이 제한적인 경우에도 앱이 자연스럽게 동작할 수 있도록 개발하는 것이

중요하다. 또한, 어떤 하드웨어에 의존하거나 의존하지 않아야 하는지를 고려하는 것도 매우 중요하다(예를 들어, 어떤 기기는 카메라, GPS 센서, 혹은 키보드가 없다).

제한된 컴퓨팅 자원

모바일 환경에서 동작하는 앱 개발은 데스크톱 앱 개발과 다르다. 데스크톱이나 서버와 달리 모바일 기기는 CPU 처리속도와 메모리가 제한적이다. 모바일 기기 사용자들은 사용자의 동작을 방해하거나(UI가 멈춰있는 경우), 배터리 소모가 많거나, 기기 동작 오류가 생기는 앱을 참으면서 사용하지 않는다.

안드로이드 개발자 도구(ADT)를 사용하는 개발 프로세스

ADT 개발은 표준 안드로이드 플랫폼과 함께 구글이 관리한다. 하지만 이 둘은 전혀 다른 방식으로 관리되는데 오픈 소스적 특성이 많이 다르기 때문이다. ADT는 주요 플랫폼을 관리하는 그룹과는 다른 별도의 그룹이 개발에 참여한다. ADT는 표준 SDK와는 별개로 공개되고, 플랫폼 공개와 밀접한 관련이 있긴 하지만 원칙적으로 공개주기는 독자적이다.

표준 안드로이드 플랫폼은 비공개로 개발되는데, 개발 중인 코드 베이스에는 외부 코드를 반영하지 않는다. 제조사와 참여 회사들에 운영체제가 제공된 이후 임의의 시점에 소스 코드가 공개된다.

ADT 웹사이트⁰¹의 첫 문장에서 이미 이 프로젝트가 다르다는 점이 다음과 같이 명시되어 있다. “안드로이드 개발자 도구(ADT)는 완전히 공개된 방식으로 개발되고

01 <http://tools.android.com/>

있으며 외부 코드도 반영하고 있다.” ADT는 [Git repositories](#)⁰²와 공개된 버그 트래커 [bug tracker](#)⁰³를 통해 접근할 수 있는 여러 개의 오픈 소스 프로젝트들이 결합된 방식으로 개발된다. 관리 그룹은 커뮤니티의 참여를 장려하며 이들의 제안을 현재 릴리스에 반영하기도 한다. 프로젝트에 참여하고 싶은 경우에는 ADT 웹사이트에서 관련 정보를 참고하면 된다.

여러 운영체제에서 앱 개발하기

안드로이드가 여러 기기에서 동작하도록 설계된 것처럼, 안드로이드 앱 개발도 다양한 환경에서 할 수 있다. 이 책의 예제들은 윈도우 7 64bit와 Mac OS X를 기반으로 가정하며 이클립스Eclipse에서 만들었다. 책의 한 장은 그래들Gradle 기반의 안드로이드 스튜디오Android Studio를 소개하는 데 할애했다. 개발도구들은 다양한 플랫폼에서 동작하도록 제작되었기 때문에 대부분 운영체제와 통합개발환경(IDE)에서 예제를 따라 하는 데 지장이 없을 것이다. 또는, IDE을 전혀 사용하지 않고 안드로이드 앱을 개발할 수도 있는데, 콘솔 명령으로도 대부분 개발도구들을 사용할 수 있다.

02 · <https://android.googlesource.com/>

03 · <http://b.android.com/>

역자 서문

약간 유명한 누군가가 ‘소프트웨어가 세상을 먹어치우고 있다’고 했는데, 안드로이드는 확실히 세상을 먹어치우고 있는 소프트웨어 중의 하나다. 2014년 상반기 기준 전 세계 모바일 운영체제 점유율 70%에 육박한다. 소프트웨어 산업에서 안드로이드는 가장 중요한 플랫폼 중 하나가 되었다. 데스크톱에서는 웹 서핑이 주요한 활동이었는데, 스마트폰과 태블릿에서는 하루 평균 사용시간 2시간 반 중 2시간 동안 앱을 활용한다. 즉, 모바일에서는 데스크톱과는 달리 앱의 비중이 절대적이다.

안드로이드 앱 개발은 많은 지식이 한꺼번에 결합되어야 하는 작업이다. 화면에 정적으로 보여지는 요소만으로도 구성될 수 있는 웹 페이지와는 달리 모바일 앱은 거의 반드시 기능적으로 동작해야 한다. 이 책에서 다루는 안드로이드 개발자 도구(ADT)는 안드로이드 플랫폼의 확산에 매우 큰 역할을 했다고 생각한다. ADT를 어떻게 활용하느냐에 따라 앱의 품질이 상당히 차이가 날 수 있다. 안드로이드 개발자 웹사이트에서 상세하게 소개하지만, 이 책에서는 아주 체계적으로 ADT를 활용하는 방법을 제시하고 있어 안드로이드 앱을 개발하는 사람이라면 반드시 읽어 봐야 할 책이라고 생각한다.

예제 파일

예제 코드는 <https://github.com/mwolfson/ToolsDemo>에서 내려받을 수 있다.

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. 무료로 업데이트되는, 전자책 전용 서비스입니다.

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

지금 보시는 전자책에 소유권한을 표시한 문구가 없거나 타인의 소유권한을 표시한 문구가 있다면 위법하게 사용하고 있을 가능성이 높습니다. 이 경우 저작권법에 의해 불이익을 받으실 수 있습니다.

다양한 기기에 사용할 수 있습니다. 또한 한빛미디어 사이트에서 구입하신 후에는 횡수에 관계없이 다운받으실 수 있습니다.

한빛미디어 전자책은 인쇄, 검색, 복사하여 붙이기가 가능합니다.

전자책은 오타자 교정이나 내용의 수정·보완이 이뤄지면 업데이트 관련 공지를 이메일로 알려드리며, 구매하신 전자책의 수정본은 무료로 내려받으실 수 있습니다.

이런 특별한 권한은 한빛미디어 사이트에서 구입하신 독자에게만 제공되며, 다른 사람에게 양도나 이전은 허락되지 않습니다.

차례

01	코드 테스트	1
	1.1 Logcat.....	1
	1.2 디버깅.....	16
	1.3 Lint.....	25
02	이벤트 시뮬레이션	36
	2.1 위치와 경로 시뮬레이션.....	36
	2.2 전화 관련 시뮬레이션.....	39
	2.3 네트워크 파라미터 변경.....	42
	2.4 센서 에뮬레이션을 기기에 적용하기.....	43
	2.5 센서 고급 테스트.....	44
	2.6 개발자 옵션 메뉴.....	48
03	빌드 도구	51
	3.1 코드 컴파일하기.....	51
	3.2 배포용 APK 패키징.....	52
	3.3 Ant로 콘솔에서 빌드하기.....	58
	3.3.1 프로젝트 설정.....	59
	3.4 앱 패키징 고급 기능 활용.....	65
	3.5 Gradle 기반 빌드 도구.....	75

0 4	시스템 리소스 모니터링	83
	4.1 안드로이드의 메모리 사용.....	83
	4.2 Dalvik Debug Monitor Server (DDMS).....	84
	4.3 Memory Analyzer Tool (MAT).....	93
0 5	유저 인터페이스 작업하기	100
	5.1 Android Layout Basic Concepts 안드로이드 레이아웃의 기본 개념.....	100
	5.2 XML 파일 직접 편집하기.....	109
	5.3 그래픽 작업하기.....	120
0 6	그래픽 편집기 사용하기	127
	6.1 Graphical Layout 편집기를 사용한 레이아웃 생성.....	127
	6.2 Palette.....	129
	6.3 Canvas.....	131
	6.4 개요(Outline) 보기.....	135
	6.5 속성 편집기	135
	6.6 Configuration Chooser 설정 선택.....	136

7.1 UI 성능의 개념.....	141
7.2 Hierarchy Viewer	143
7.2 Hierarchy Viewer 시작하기.....	144
7.3 Lint로 문제 해결하기	159
7.4 Monkey 앱 실행기.....	161
7.5 Monkeyrunner.....	163
7.6 감사합니다!	166

1 | 코드 테스트

1.1 Logcat

안드로이드 플랫폼은 시스템 정보를 수집하고 표시하기 위해 logcat이라는 로그 메커니즘을 제공한다. 시스템과 여러 앱에서 수집된 로그는 버퍼 형태로 출력되며, logcat 명령어로 출력을 선별할 수 있다. Log4J 또는 java.util.logging 패키지를 사용한 경험이 있다면 logcat이 이들과 매우 유사하다고 느낄 것이다. 여러 시스템의 출력을 한 곳에서 볼 수 있고, 특정한 앱과 관련 있는 정보만을 선택해서 볼 수도 있다. logcat은 개발을 훨씬 편리하게 해주므로 logcat의 선택사항을 잘 이해하는 것이 좋다.

안드로이드는 공통 로그 파일에 시스템의 거의 모든 사항을 기록한다. 가비지 콜렉션(garbage collection), 시스템의 다양한 활동, 앱 출력과 같은 정보는 모두 같은 파일에 기록된다. 즉, 광범위한 정보를 한 곳에 모두 기록한다. 이 파일은 기기에 설치된 모든 파일에서 공유된다는 점도 명심하자. 따라서, 중요한 정보는 로그에 기록되지 않도록 해야 하는데, Proguard 유틸리티를 사용해 코드 난독화를 수행하고 특정한 세부사항을 숨길 수 있다. 이 유틸리티를 사용하면 배포용 앱을 패키징할 때 로그 출력 구문을 제거할 수도 있다.

1.1.1 Viewing the Logcat File Logcat 파일 보기

log 파일 전체를 보려면(필터 선택 없음), 다음 명령을 실행한다.

```
adb logcat
```

이 출력은 아주 상세한 로그를 모두 출력하며, 로그에는 시스템의 모든 프로세스

정보가 있다.

1.1.2 Anatomy of a Log Message 로그 메시지 구조

각 로그 메시지는 출력 시 필터 적용이 가능한 다양한 메타데이터를 포함한다.

Log level

앱의 관점에서 메시지의 중요도를 표시한다.

Log tag

메시지에 연결된 프로세스나 ID를 정의한다.

Log message

메시지의 내용

Logcat 출력 읽기

logcat의 각 줄은 중요한 정보를 여러 포함하는데, 관련 정보를 찾는 방법을 알아 보자.

다음은 logcat 파일 내 기록된 몇 가지 예다.

```
E/PowerManagerService( 170): Excessive delay setting brightness: 101ms,
mask=2
V/PhoneStatusBar( 308): setLightsOn(true)
I/ActivityManager( 170): No longer want com.android.contacts (pid 598):
hidden
I/ActivityManager( 170): Displayed com.tools.demo/.LogcatDemoActivity:
+955ms
D/UI ( 897): The user has pressed the button
```

```
D/UI ( 897): The user entered a value: value from the call is: 24324
```

앞의 예 중 하나를 선택해 어떤 정보가 기록되었는지를 이해해 보자. 첫 번째 구문은 시스템에서 생성된 것으로(시스템과 사용자 정의 메시지는 모두 같은 파일에 기록된다는 점을 명심하자), OS와 액티비티의 상호작용을 관리하는 안드로이드 컴포넌트의 출력이며 Activity Manager가 LogcatDemoActivity를 실행하는 데 얼마나 시간이 걸렸는지를 기록했다.

```
I/ActivityManager( 170): Displayed com.tools.demo/.LogcatDemoActivity:  
+955ms
```

이 구문을 좀 더 잘 이해하기 위해 개별 항목을 아래서 살펴보자.

/

메시지의 중요도를 표시한다('1.1.3 로그 레벨 필터링' 참조). 이 로그는 info 레벨이라는 것을 의미한다.

ActivityManager

이 태그는 로그 메시지를 생성할 때 사용된다('1.1.4 태그 출력 필터링' 참조). 로그 메시지를 생성한 시스템이 어떤 것(여기서는 Activity Manager다)인지를 알려준다.

(170)

이 메시지를 생성한 앱의 프로세스 ID다. 앱 실행 시 부여되는 고유한 식별자이며 메시지를 선택하는 좋은 기준이 된다.

```
Displayed com.tools.demo/.LogcatDemoActivity: +955ms
```

이 메시지는 사용자가 정의한 내용이다. 이 메시지는 액티비티가 언제 시작했으며 생성되는 데 시간이 얼마나 걸렸는지를 알려준다. 이 로그는 앱에 입력된 텍스트가 출력된 것이다.

다른 항목을 살펴보자. 이번에는 필자가 직접 코드에 작성한 내용이 출력되었다. 직접 작성한 로그 메시지도 시스템 메시지와 똑같은 정보와 메시지를 담고 있음을 알 수 있다.

```
D/UI ( 897): The user entered a value: value from the call is: 24324
```

사용자 정의 로그도 시스템 메시지와 완전히 동일한 구조를 가진다.

D

이 메시지는 Debug 중요도 레벨이다.

UI

필자가 정의한 사용자 상호작용 이벤트 기록 태그.

(897)

OS에서 앱에 부여한 ID.

The user entered a value: value from the call is: 24324

사용자가 양식에 입력한 24324라는 값을 표시.

logcat에는 많은 정보가 담겨있어서 관리가 어려울 수 있다. 다음으로는 작업을 좀 더 수월하게 만드는 log를 생성과 선별하는 전략을 논의한다.

1.1.3 로그 레벨 필터링

메시지의 중요도에 따라 로그를 선택할 수 있다. 로그 메시지는 디버그 우선순위에 따라 표시된다. 최저 레벨을 선택할 수 있는데, 선택된 최저 레벨 이상으로 지정된 메시지만 표시된다.

로그 레벨의 차이를 아는 것이 중요하다. 보고자 하는 메시지의 종류에 따라 적절한 레벨이 선택되었는지를 확인해야 한다. 로그 레벨에 대해서는 표 1-1을 참고하라.

표 1-1 중요도 순의 로그 레벨

식별자(ID)	이름	우선순위
V	Verbose (모두 표시)	1 (가장 낮음)
D	Debug	2
I	Info	3
W	Warning	4
E	Error	5
F	Fatal	6
S	Silent (표시 없음)	7

특정 레벨의 모든 메시지(지정된 레벨 이상의 우선순위를 모두 포함)를 보려면 다음을 입력한다.

```
adb logcat *:Identifier (ID)
```

예를 들어, Error 이상의 우선순위를 모두 보려면 다음과 같이 입력한다.

```
adb logcat *:E
```

NOTE

앱에서는 D 레벨로 로그를 지정하는 것이 좋다. 안드로이드 Log API에서는 'Debug 로그는 컴파일 시 포함되지만 런타임에서 제거된다(Debug logs are compiled in but stripped at runtime)'고 명시되어 있다. 이 의미는 D 레벨로 생성된 로그는 배포용 앱에서 출력되지 않는다는 의미이다. 따라서, 민감한 데이터가 실수로 기록되는 것을 방지하려면 이 로그 레벨을 사용하는 것이 가장 안전하다.

1.1.4 태그 출력 필터링

logcat에서 필터 표현(filter expression)을 적용하면 원하는 메시지만 골라낼 수 있다. 필터 표현은 tag:level 형식으로 되어 있다. 원하는 메시지만 골라내려면 한 개 이상의 필터 표현을 적용하면 된다.

원하는 메시지를 찾으려면 자바 코드에서 직접 태그를 정의해야 하는데, 문법은 다음과 같다.

```
Log.level("CustomTag", "Log message")
```

사용 예는 다음과 같다.

```
Log.D("UI", "User entered a value: " + myEditText.getText());
```

태그된 메시지만을 보려면 다음 명령을 사용한다.

```
adb logcat UI:D *:S
```

1.1.5 Logcat 좀 더 잘 사용하기

앞에서 살펴보았듯이, logcat 출력을 선택하는 방법은 여러 개여서 원하는 메시지를 보고자 할 때 편리하다. logcat 명령에 필터 표현을 필요한 만큼 지정하면 표시되는 메시지를 원하는 수준으로 조정할 수 있다.

필터 표현을 여러 개 적용하려면, 다음과 같이 logcat 명령에 필터 표현을 추가하면 된다.

```
adb logcat TAG1:level
          TAG2:level
          TAG3:level
```

예를 들어, Activity Manager의 모든 통계 수치를 보려면 ActivityManager:* 태그를 사용한다. Power Manager 컴포넌트의 Error 이상의 중요도 레벨 메시지만 보려면 PowerManagerService:E 태그를, 앱에서 정의된 User Interface 태그를 보려면 UI:*을 사용한다. 그 밖의 메시지는 *:s(다른 메시지는 출력하지 않음)를 사용해 출력되지 않도록 하자. 이 필터를 합치면 다음과 같이 명령을 작성할 수 있다.

```
adb logcat ActivityManager:* PowerManagerService:E UI:D *:s
```

1.1.6 대체 로그 버퍼 보기

로그 시스템은 로그 메시지에 여러 개의 버퍼를 사용한다. 특정한 콘텐츠(무선이나 이벤트와 같은 경우)의 출력은 기본 버퍼 대신 대체 버퍼에 남겨진다. 추가 로그 메시지를 보려면, logcat을 -b 옵션으로 시작하여 보고자 하는 대체 버퍼를 지정하면 된다. 예를 들어, 무선 버퍼를 보려면 다음과 같이 입력한다.

```
adb logcat -b radio
```

1.1.7 지정된 출력 포맷

로그 메시지에는 레벨, 시간, 프로세스 ID, 앱, 태그, 오류 텍스트와 같은 다양한 메타데이터 필드가 있다. 특정 메타데이터 필드를 표시하고자 할 때 사용하는, 정의된 출력 양식이 있다.

brief

태그와 프로세스의 PID를 표시

raw

다른 메타데이터는 제외하고 로그 메시지 자체만 표시

time

날짜, 시간, 태그, PID만 표시

long

모든 메타데이터 필드를 표시하고 각 메시지 사이에 공백 줄을 삽입

예를 들어, 시간 포맷을 출력하려면 다음과 같이 입력한다.

```
adb logcat -v time
```

1.1.8 이클립스 Logcat Viwer

이클립스의 표준 Java perspective에서 그림 1-1와 같이 화면 하단의 여러 개의

탭 중 logcat 탭(📄)을 볼 수 있다. 이 도구로 추가 UI를 사용해 현재 연결된 기기의 logcat을 탐색할 수 있다.

NOTE

시스템 로그 파일은 메시지가 너무 많을 수 있다. 좀 더 수월하게 메시지를 보려면 원치 않는 공통 시스템 정보를 제거하는 제거 필터를 사용하면 되는데, 다음과 같이 제거 필터를 작성하면 된다.

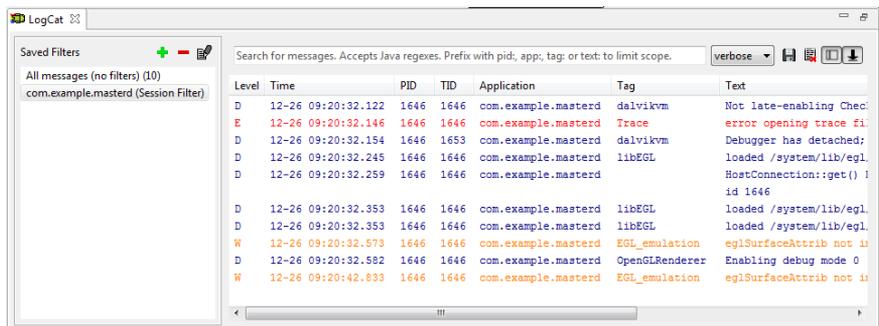
1. 새 필터를 생성하는 더하기 아이콘(+)을 선택.
2. 'Filter Name'란에 필터 이름을 지정.
3. 'by Log Tag'에 필터값을 다음과 같이 입력한다.

```
^(?!exclude - term1|excludeterm2|excludeterm3).* $
```

4. 필자는 다음의 필터 양식을 기준으로 많이 사용한다.

```
^(?!dalvikvm|ActivityManager|SystemServer|BackupManagerService).* $
```

그림 1-1 Logcat 도구



1.1.9 Logcat 예시

Logcat은 기능이 강력하지만, 공통 로그에 출력되는 정보의 양이 많아서 원하는 정보를 찾기가 어렵다. 많은 정보가 담긴 시스템 로그에서 원하는 정보를 선택하는 방법을 간단한 예를 통해 알아보면서 코드 출력을 이해해 보자.

이 예는 사용자에게 미국 달러화 값을 입력 받아서 유로화로 변환해 주는 웹 서비스에 보내는 코드를 보여주는데, 값을 지정된 양식으로 변환해 사용자 인터페이스에 표시한다.

로그 영역 정하기

로그를 생성하는 코드의 특징을 살펴보는 것이 중요하다. 로직을 별개의 영역(사용자 정의 태그로 표시)으로 분리해야 하는데, 기능에 따라 로그를 구별할 수 있어 별개의 영역으로 분리하면 특정 영역에 집중하기 쉽기 때문이다. 이 경우 독자적으로 추적해야 하는 특정한 기능 영역이 있는데, 살펴볼 영역은 다음과 같다.

User Interface

사용자가 입력한 값과 실제로 표시되는 값 확인, 버튼이 눌린 경우 확인 등.

Network

전송하는 URL 확인, 연결 오류 확인, 요청/응답 값 확인 등.

JSON 파싱

JSON 파싱 과정에서 JSON 값 출력 등.

Formatting

앱에 사용되는 값을 포맷팅하는 알고리즘 확인.

AsyncTasks

이 코드의 생명주기 호출 추적.

로그 구문 생성

로그를 출력할 분류를 정하고 나면 각 분류의 태그를 생성한다. 지금 경우에는 다음과 같이 한다.

- UI
- NETWORK
- JSON
- FORMAT

NOTE_

자주 사용하는 카테고리는 유틸리티 클래스의 상수값으로 생성한다(예, `public static final String TAG_UI = "my_tag_ui"`). 그런 다음 코드에서 태그를 사용한다(예, `Log.d(LogUtil.TAG_UI, "UI Log message")`). 이 방법으로 여러 개의 액티비티에 걸쳐 있는 특정한 하위 시스템을 필터로 선별해 확인할 수 있다.

그런 다음, 로그 구문을 코드에 삽입해 앞에서 정의한 태그를 적절히 사용한다. 다음의 코드는 Web, JSON, 포매팅 기능에서 로그를 사용하는 예다.

```
public static String ConvWebCall(String amount) {  
    // To save space, I removed code not important to the example  
  
    try {  
        HttpClient client = new DefaultHttpClient();  
        HttpGet request = new HttpGet();
```

```

    ...

    sb.append(amount);
    Log.d("NETWORK", "The URL we are sending is: " + sb.toString());

    request.setURI(new URI(sb.toString()));
    HttpResponse response = client.execute(request);
    Log.d("NETWORK", "Response received");
    ...

    Log.d("NETWORK", "The return String is: " + retStr.toString());
    String page = retStr.toString();
} catch (URISyntaxException e) {
    Log.e("NETWORK", "URISyntaxException", e);
    e.printStackTrace();
    ...
}
Log.d("NETWORK", "Return value is: " + page);
return page;
}

public static String parseConvedValue(String page) {
    String curr = "";
    try {
        JSONObject jso = new JSONObject(page);
        Log.d("JSON", "JSON Value: " + jso);
        curr = jso.optString("v");
    } catch (JSONException e) {
        Log.e("JSON", "Parsing exception: ", e);
    }
    Log.d("JSON", "JSON Value: currency element is " + curr);
    return curr;
}
}

```

```

private String formatEuroForDisplay(String amount, String name) {
    String euro = "Euro 00.00";
    Log.d("FORMAT", "Before formatEuroForDisplay: " + amount);
    if (amount != null) {
        int index = amount.indexOf(".");
        String euros = amount.substring(0, index + 3);
        Log.d("FORMAT", "Euros back from NETWORK: " + amount);
        euro = "Euro " + euros;
    }
    Log.d("FORMAT", "After formatEuroForDisplay: " + euro);
    String euroString = euro;
    return euroString;
}

```

상세 로그

출력을 다 보려면 필터 없이 logcat을 실행하면 된다(`adb logcat` 명령). 아래의 출력 결과가 보여주듯이 이 출력은 정보가 많아서 이해하기 어렵다. 아래는 환율 변환 작업을 한번 수행시켰을 때 발생하는 출력인데, 태그를 보면 프로세스가 어떻게 진행되는지 볼 수 있다. 처음에는 UI 메시지, 그리고는 FORMAT, NETWORK 등으로 이어진다.

```

I/ActivityManager( 170): START {cmp=com.tools.demo/.LogcatDemoActivity
u=0} from pid 897
W/WindowManager( 170): Failure taking screenshot for (328x583) to layer
21010
D/dalvikvm( 170): WAIT_FOR_CONCURRENT_GC blocked 0ms
D/dalvikvm( 170): GC_EXPLICIT freed 114K, 39% free 13611K/22023K,
paused 10ms+9ms, total 267ms
I/Choreographer( 897): Skipped 35 frames! The application may be doing

```

too much work on its main thread.

```
E/PowerManagerService( 170): Excessive delay setting brightness: 101ms,
mask=2
```

```
V/PhoneStatusBar( 308): setLightsOn(true)
```

```
I/ActivityManager( 170): No longer want com.android.contacts (pid 598):
hidden
```

```
I/ActivityManager( 170): Displayed com.tools.demo/.LogcatDemoActivity:
+955ms
```

```
D/UI ( 897): The user has pressed the button
```

```
D/UI ( 897): The user entered a value: value from the call is:
```

```
24324
```

```
D/FORMAT ( 897): formatForWebcall() before: 24324
```

```
D/FORMAT ( 897): formatForWebcall() after: 243.24
```

```
D/ASYNC ( 897): onPreExecute()
```

```
D/NETWORK ( 897): URL to send: http://rate-exchange.appspot.com/
currency?from=USD&to=EUR&q=243.24
```

```
D/dalvikvm ( 897): GC_CONCURRENT freed 121K, 2% free 11063K/11271K,
paused 17ms+32ms, total 78ms
```

```
D/NETWORK ( 897): Response received
```

```
D/NETWORK ( 897): The return String is: {"to": "EUR", "rate":
0.756258035, "from": "USD", "v": 183.9522044334}
```

```
D/ASYNC ( 897): doInBackground(): {"to": "EUR", "rate":
0.756258035, "from": "USD", "v": 183.9522044334}
```

```
D/JSON ( 897): Json Object is:
```

```
{"to": "EUR", "v": 183.9522044334, "from": "USD", "rate": 0.756258035}
```

```
D/JSON ( 897): Json parsed currency value: 183.9522044334
```

```
D/ASYNC ( 897): onPostExecute(): result is: 183.9522044334
```

```
D/FORMAT ( 897): Before formatEuroForDisplay: 183.9522044334
```

```
D/FORMAT ( 897): Euros back from call: 183.9522044334
```

```
D/FORMAT ( 897): After formatEuroForDisplay: Euro 183.95
```

```
D/UI ( 897): Setting value on screen to: Euro 183.95
```

```
I/ActivityManager( 170): START {act=android.intent.action.MAIN cat=[an
```

```
droid.intent.category.HOME] flg=0x10200000
cmp=com.android.launcher/com.android.launcher2.Launcher u=0} from pid 170
  W/WindowManager( 170): Failure taking screenshot for (328x583) to layer
21015
  W/IInputConnectionWrapper( 897): showStatusIcon on inactive
InputConnection
  I/Choreographer( 489): Skipped 42 frames! The application may be doing
too much work on its main thread.
```

logcat 필터링하기

필자는 전체 프로세스를 살펴보려고 종종 필터를 적용하지 않은 출력을 보기도 하지만, 많은 경우에는 좀 더 상세하게 선택된 출력이 필요하다. 이를 위해 보고자 하는 특정 분류에 해당하는 사용자 정의 태그를 만든다. 관심 있는 정보만을 출력하기 때문에 특정 작업의 이해가 쉬워진다.

예를 들어, 사용자 상호작용에만 관심 있는 경우 UI 태그를 기준으로 필터를 적용한다. 이 경우, UI 태그 옆에 * 기호를 붙이고(모든 메시지 선택), *.s를 지정해 다른 메시지는 나오지 않도록 한다.

```
$ adb logcat UI:* *:s
D/UI    ( 897): The user has pressed the button
D/UI    ( 897): The user entered a value: value from the call is: 24324
D/UI    ( 897): Setting value on screen to: Euro 183.95
```

다른 예로는 웹 호출과 응답 해석이 있는데, NETWORK과 JSON 분류를 사용해 해당 로직을 살펴보자.

```
$ adb logcat JSON:* NETWORK:* *:s
```

```
D/NETWORK ( 897): URL to send: http://rate-exchange.appspot.com/currency?
from=USD&to=EUR&q=243.24-->
D/NETWORK ( 897): Response received
D/NETWORK ( 897): The return String is: {"to": "EUR", "rate": 0.756258035,
"from": "USD", "v": 183.9522044334}
D/JSON ( 897): Json Object is:
{"to": "EUR", "v": 183.9522044334, "from": "USD", "rate": 0.756258035}
D/JSON ( 897): Json parsed currency value: 183.9522044334
```

1.2 디버깅

디버깅은 개발에서 중요한 단계이고 어떤 경우에는 코드 작성보다 시간이 더 많이 걸릴 수도 있는데, 안드로이드 앱 디버깅은 OS와 연동된 하위 시스템이 많아서 특히 더 까다롭다. ADT 도구는 통합 디버깅 환경을 제공해 이 과정을 쉽게 해 준다.

디버깅을 하는 가장 흔한 방법은 특정한 실행 경로로 코드가 실행될 때 발생하는 중단점(breakpoint)를 삽입하는 것이다. 프로그램 실행이 이 지점에서 멈추는데, 이로 시스템의 상태를 살펴볼 수 있다(현재 변수 값과 앱 상태를 포함). 이 정보를 활용해 코드 동작을 분석하고 오류를 찾으면 된다.

1.2.1 앱 디버그 설정

앱을 디버그하려면 앱 매니페스트에서 앱을 디버그 가능(debuggable)으로 설정해야 한다. (이클립스나 다른 IDE의) ADT 도구를 사용하고 있다면 자동으로 디버그 가능으로 설정되지만, 이 도구를 사용하지 않고 프로젝트를 디버그하려면 이 값을 수동으로 설정해야 한다. 앱을 디버그 가능하게 수동으로 설정하려면 AndroidManifest.xml 파일에서 android:debuggable="true"로 속성을 지정하면 되고 설정은 다음과 같다.

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:debuggable="true">
```

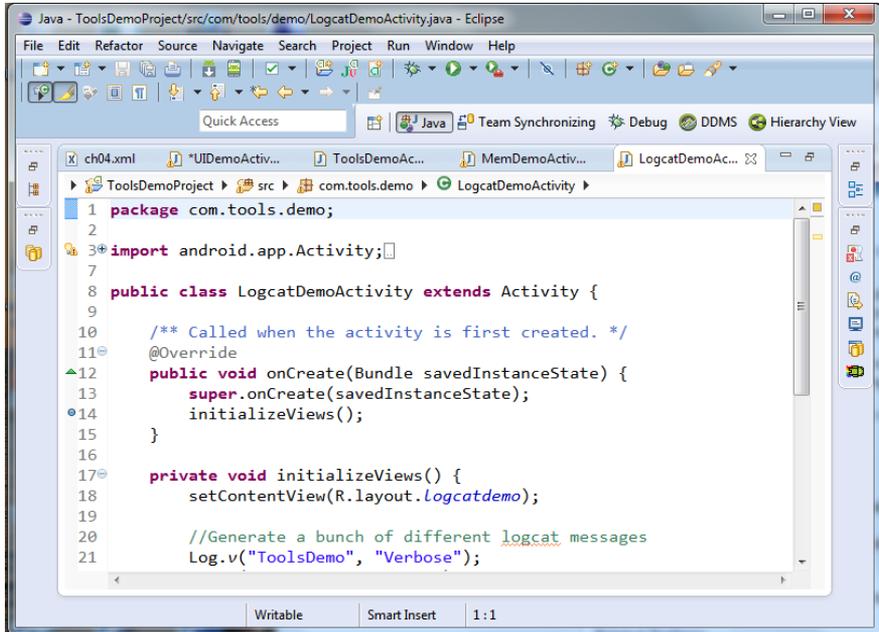
NOTE_

디버그 플래그를 수동으로 설정한 경우 앱 배포 시에는 디버그 플래그를 제거해야 한다. 제거하지 않으면 성능에 부정적 영향을 미치므로 배포용에서는 제거하는 것이 좋다. 제거되지 않은 경우 Link checker(7.3 Lint 참조)가 릴리스 빌드 시 이 속성이 설정되어 있다는 경고를 표시한다.

1.2.2 디버그 지점 설정

소스 코드에 중단점을 설정하는 것부터 디버깅을 시작하는 것이 일반적이다. 이클립스에서는 코드를 클릭해서 ‘Toggle Breakpoint’을 선택하는데, 윈도우 또는 리눅스에서는 Ctrl+Shift+B, 맥에서는 Command+Shift+B를 눌러 중단점을 설정할 수도 있다. 중단점을 설정한 다음 실행하면, 실행 중 특정 코드에 도달하면 IDE가 ‘Debug perspective’로 전환된다. 예를 들어, 그림 1-2에서는 LogcatDemoActivity의 onCreate() 메소드 안에 중단점을 설정했다. 코드 실행 중 이 부분의 코드 경로가 실행되면(지금의 예에서는, initializeView() 메소드에 도달하면) Debug perspective가 자동으로 시작되고, 코드 실행은 해당 구문에서 멈춘다.

그림 1-2 디버그 중단점 설정



1.2.3 이클립스 Debug Perspective

중단점을 삽입하면 Debug perspective가 자동으로 시작되는데, 수동으로 Window → Open Perspective → Debug로 실행할 수 있다. 그림 1-3은 Debug perspective의 화면이다. 화면의 몇몇 부분을 자세히 살펴보자.

Debug

현재 디버그가 실행되는 안드로이드 앱과 동작하는 스레드를 표시한다.

Variables

코드 실행 중 특정 중단점에서의 변수값을 표시한다.

Breakpoints

앱에 현재 설정된 모든 중단점 목록이다. 여기서 중단점의 활성화 또는 비활성화 등을 포함한 중단점 조작이 가능하다.

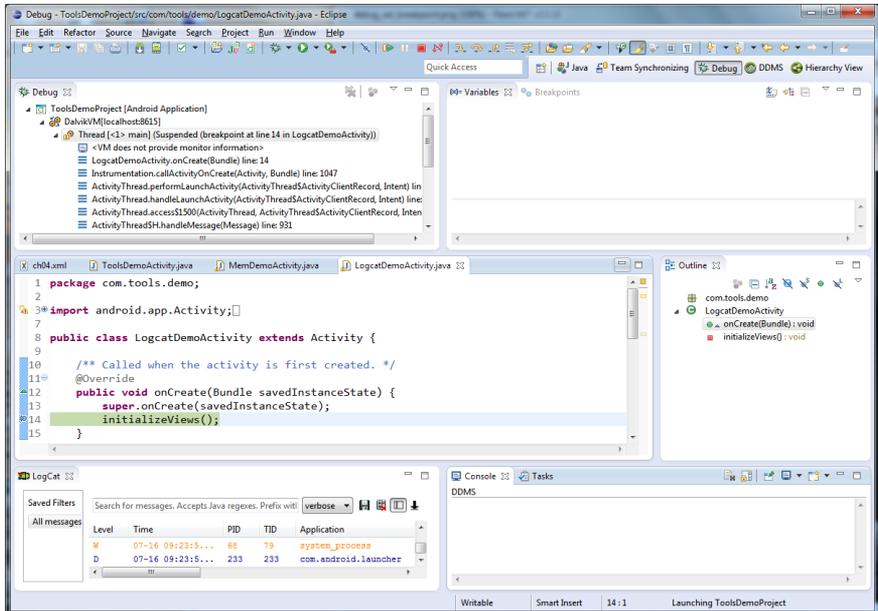
Logcat

시스템 로그 메시지를 표시한다.

Code and outline tabs

현재 실행되는 소스 코드와 개요를 보여준다.

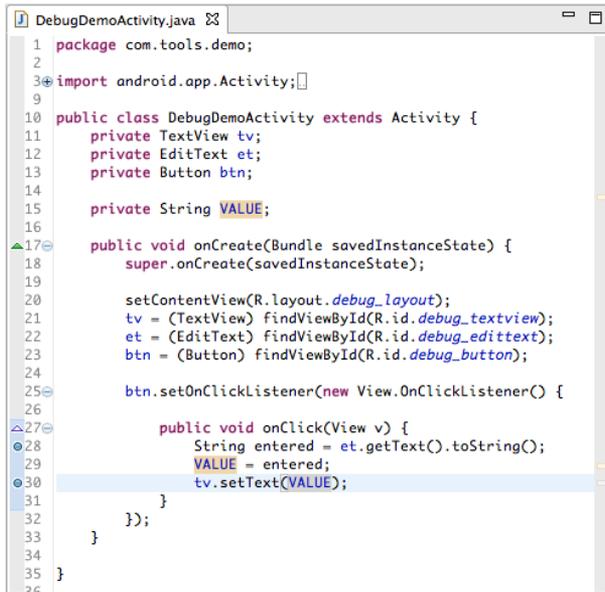
그림 1-3 Debug perspective



1.2.4 디버깅의 예

코드의 특정 요소를 어떻게 디버깅하는지 예를 통해 살펴보자. 디버거를 사용해서 실행 사이클의 다른 지점의 값을 어떻게 살펴볼 수 있는지 설명하고자 그림 1-4의 매우 단순한 코드를 만들었다. 이 코드는 사용자에게 값을 입력 받아 내부 변수로 저장하고 버튼이 눌리면 다시 그 값을 표시해 주며, 앱 동작은 그림 1-5와 같다.

그림 1-4 디버그 예의 소스 코드



```
1 package com.tools.demo;
2
3 import android.app.Activity;
4
5
6
7
8
9
10 public class DebugDemoActivity extends Activity {
11     private TextView tv;
12     private EditText et;
13     private Button btn;
14
15     private String VALUE;
16
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19
20         setContentView(R.layout.debug_layout);
21         tv = (TextView) findViewById(R.id.debug_textview);
22         et = (EditText) findViewById(R.id.debug_edittext);
23         btn = (Button) findViewById(R.id.debug_button);
24
25         btn.setOnClickListener(new View.OnClickListener() {
26
27             public void onClick(View v) {
28                 String entered = et.getText().toString();
29                 VALUE = entered;
30                 tv.setText(VALUE);
31             }
32         });
33     }
34 }
35
36
```

디버그 지점 설정

앱 디버깅의 첫 번째 단계는 적절한 디버깅 지점을 결정하는 일이다. 예를 들어, 두 실행 지점에 있는 VALUE 변수의 값을 알고 싶다고 하자. 첫 번째는 사용자가 무언가를 입력하기 전으로 줄 28에 해당한다. 변수에 값이 할당된 다음에도 확인하고 싶어서 줄 30에 디버그 지점을 하나 더 설정했다. 중단점 설정은 줄 번호에서 오른쪽 쪽 클릭을 한 다음 'Toggle Breakpoint'를 선택하면 된다. 그림 1-4에 보면 각 줄

옆의 작은 파란색 표시가 있는데, 이는 해당 지점에 중단점이 설정되었다.

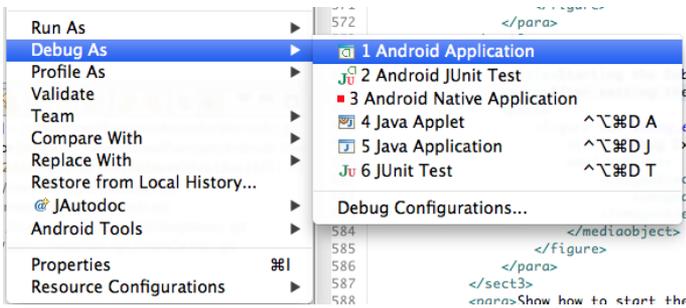
그림 1-5 디버깅 예의 실행 결과



디버거 실행

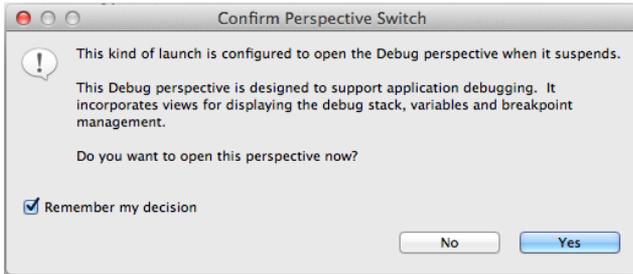
디버거 실행은 디버그 지점을 설정한 뒤 그림 1-6과 같이 프로젝트에서 오른쪽 클릭으로 Debug As → Android Application 선택하면 된다.

그림 1-6 오른쪽 클릭 메뉴로 디버깅 실행하기



이 과정은 앱을 실행하는 것과 다름없다. Android Device Chooser에서 기기를 선택해야 하고, 디버깅을 시작할 지점으로 앱 화면을 이동해야 한다. 해당 지점에서 그림 1-7처럼 Debug perspective 실행 여부를 묻는 대화창이 화면에 표시 되는데, Yes(그리고 밑에 있는 선택항목에서 다음에는 자동으로 수행되도록 선택)를 클릭한다. 그러면 Debug perspective가 열린다.

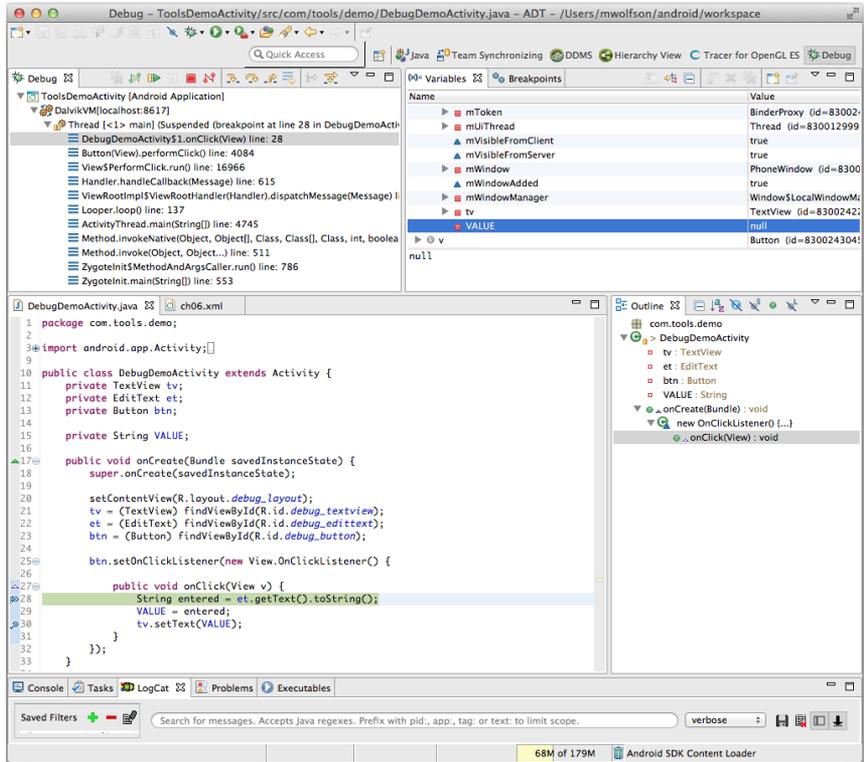
그림 1-7 디버그 확인 대화창



코드 단계별로 실행하기

그림 1-8과 같이 Debug perspective가 표시되고 디버거는 첫 번째 중단점에서 실행을 일시 정지한다. 줄 28의 첫 번째 디버그 지점이 밝게 강조 표시되었음을 확인할 수 있다. 이렇게 되면 이 지점의 코드에 대한 정보를 좀 더 자세히 알려 주는 다른 도구를 사용할 수 있는데, 그 중 Variables 탭(상단 오른쪽)을 사용해 VALUE 변수의 현재 값을 볼 수 있다. 지금은 아무것도 입력하지 않았기 때문에 값은 null이다.

그림 1-8 첫 번째 디버그 포인트



다음 디버그 지점으로 진행하려면 디버거에게 진행을 명령해야 하는데, 디버그 툴바의 Resume 버튼(▶)을 누르면 된다.

NOTE

버튼에 마우스 커서를 갖다 대면 버튼의 기능이 표시되며 좀 더 상세한 선택사항을 볼 수 있다.

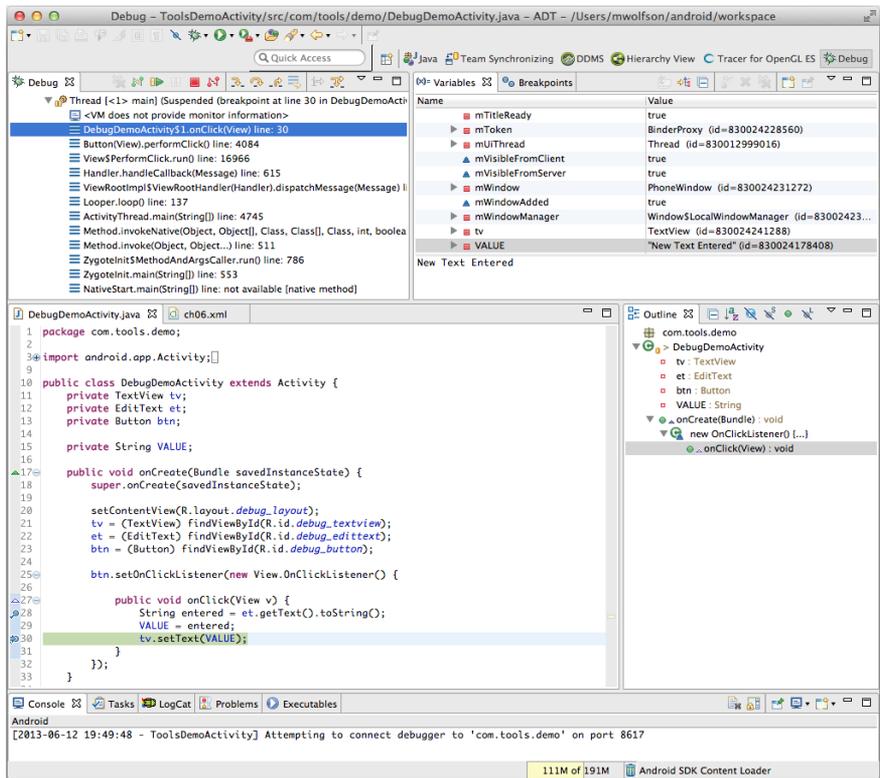
그림 1-9 디버그 툴바



Resume 버튼을 누르면 디버거는 코드를 실행하고 다음 디버그 지점(여기서는 줄 30)에서 멈춘다. 코드가 다음 디버그 지점으로 진행되었기 때문에 화면은 그림 1-10과 같이 바뀐다. 여기서 실행이 멈추므로 코드를 다시 살펴보면서 컴포넌트의 값이 어떻게 바뀌었는지 알 수 있다.

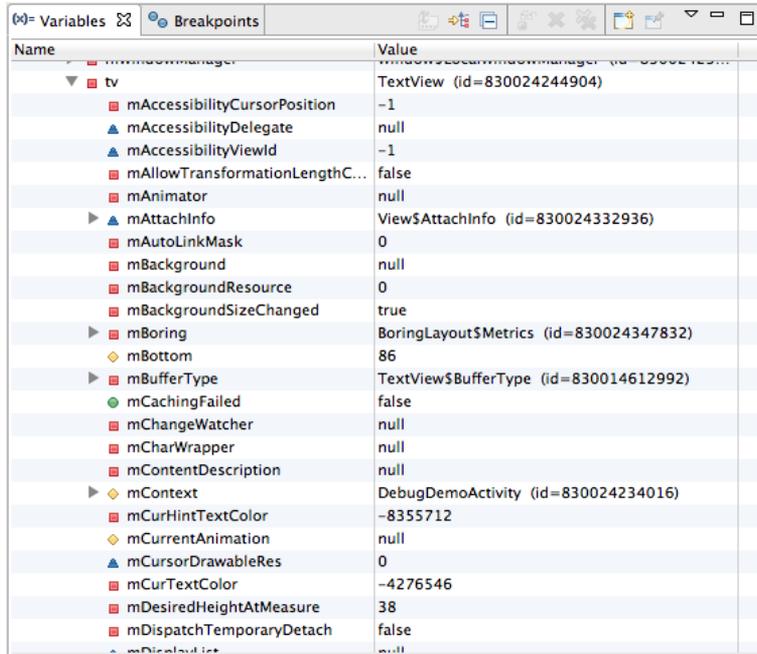
Variables 탭을 보면 VALUE 변수의 값이 'New Text Entered'(새 텍스트 입력됨)으로 바뀌었음을 알 수 있다.

그림 1-10 두 번째 디버그 포인터



이 지점의 다른 값도 살펴볼 수 있는데, 그림 1-11과 같이 Variables 탭의 tv 항목(코드에서 TextView를 의미한다)을 클릭하면 여러 정보를 볼 수 있다. 여기서는 Android 속성(패딩, 애니메이션, 또는 포매팅) 또는 상태 정보(현재 표시되는 텍스트)와 같은 정보를 얻을 수 있다. 항목의 상세 속성을 파악하고자 할 때 여러 실행 시점에서 다양한 값을 살펴보는 것도 무척 도움이 된다.

그림 1-11 다른 값들 살펴보기



1.3 Lint

Lint는 ADT 16에서 처음 도입된 정적 분석 도구로 코드를 스캔하고 잠재적인 버그를 파악한다. 명령줄에서 실행할 수도 있고 Java 또는 XML 에디터, 빌드 도구에서도 사용할 수 있다. Lint는 설정된 오류 확인 규칙과 패턴을 바탕으로 코드 내 잠

재적 문제를 확인한다. Lint는 이런 문제를 알려주고, 많은 경우 문제 해결 방법 등을 제안하거나 알려준다. 이처럼 Lint는 강력한 도구이면서 코드 품질을 최소한의 노력으로 개선할 수 있는 사용하기 쉬운 도구다.

Lin는 다양한 이슈를 확인해 주는데, 그 예는 다음과 같다.

- 접근성과 국제화; 누락된 번역 등
- 사용자 인터페이스 최적화; 사용되지 않는 뷰 강조 등
- 보안: HTTPS 미사용 강조 등
- 코드 오류: 클래스 간 비일관적인 배열 크기 등
- 리소스 문제; 특정 아이콘의 해상도 누락 등

lint --list 명령은 현재 Lint에서 설정된 모든 이슈 목록을 보여준다. 이 명령의 결과는 현재 사용되는 분류를 출력하고 각 이슈에 대한 간략한 설명을 제공한다.

```
$ lint --list
```

```
Valid issue categories:
```

```
Correctness
Correctness:Messages
Security
Performance
Usability:Typography
Usability:Icons
Usability
Accessibility
Internationalization
```

```
Valid issue id's:
```

```
"ContentDescription": Ensures that image widgets provide a
```

```
contentDescription
"LabelFor": Ensures that text fields are marked with a labelFor attribute
"FloatMath": Suggests replacing android.util.FloatMath calls with
    java.lang.Math
"FieldGetter": Suggests replacing uses of getters with direct field access
    within a class
...
```

1.3.1 명령줄 사용하기

Lint를 사용하는 가장 간단한 방법은 프로젝트에서 Lint를 실행하여 보고된 오류를 살펴보는 것인데, Lint가 찾은 오류를 전체적으로 살펴볼 수 있다.

이 도구를 실행하려면 lint 명령을 실행하고 안드로이드 소스 코드 디렉터리를 지정하면 된다. 여러 프로젝트가 들어 있는 디렉터리를 지정하면 Lint는 해당 경로의 모든 프로젝트에 대해 실행한다.

아래는 필자의 프로젝트에 적용한 Lint 보고서의 일부다. 보고 내용을 살펴보면 어떤 항목이 있는지 알 수 있다.

```
$ lint ./ToolsDemo
```

```
Scanning ToolsDemo: .....
Scanning ToolsDemo (Phase 2): .....
res/layout/gooduidemo.xml:15: Warning: Should use "sp" instead of "dp" for
text sizes [SpUsage]
    android:textSize="20dp" />
    ~~~~~

res/layout/baduidemo.xml:167: Warning: [I18N] Hardcoded string "Text will go
here", should use @string resource [HardcodedText]
    android:hint="Text will go here"
```

```

~~~~~
res/layout/baduidemo.xml:120: Warning: Duplicate id @+id/imageView2,
already defined earlier in this layout [DuplicateIds]
    android:id="@+id/imageView2"
~~~~~

res/layout/imagesdemo.xml:23: Warning: [Accessibility]
Missing contentDescription attribute on image [ContentDescription]
    >ImageView
    ^

res/layout/baduidemo.xml:89: Warning: This tag and its children can be
replaced by one >TextView/< and a compound drawable [UseCompoundDrawables]
    >LinearLayout
    ^

...
0 errors, 80 warnings
$

```

앞의 예는 일부분이라 Lint가 프로젝트에서 찾아낸 모든 오류를 보여주지는 않는다. 맨 마지막 줄을 보면 Lint가 0개의 오류를 찾았음을 알 수 있다(경고는 80개, 이 프로젝트는 매우 작은 규모다). 프로젝트에 Lint를 자주 적용해 코드 품질을 향상시키길 바란다.

확인 항목 제외

특정 항목의 오류(국제화를 지원하지 않는다면, 이와 관련된 오류 경고는 필요하지 않다)를 제외하고 싶을 때는, 명령줄에서 ‘--disable [목록 인자]’를 포함하면 된다. [목록 인자]는 제외하려는 이슈 목록의 ID이거나 분류다.

예를 들어, 국제화와 관련된 오류를 제외하고 싶고(예를 들어 여러분의 앱이 한국어만 지원한다고 하자), 특정한 ContentDescription 오류도 제외한다고 해 보자(이미지가 표시되지 않아도 괜찮은 경우라면). 이전과 같은 보고서를 생성하지만 --disable 인자를

포함하면 Lint는 이전보다 좀 더 적은 항목을 보여준다. 이전의 경고 80개에서 26개로 훨씬 줄었다. 이렇게 하면 이슈 목록을 줄일 수 있고 가장 중요한 이슈에 집중할 수 있다.

```
$ lint ./ToolsDemo --disable Internationalization,ContentDescription

Scanning ToolsDemo: .....
Scanning ToolsDemo (Phase 2): .....
res/layout/gooduidemo.xml:15: Warning: Should use "sp" instead of "dp" for
text sizes [SpUsage]
    android:textSize="20dp" />
    ~~~~~
...
0 errors, 26 warnings
$
```

--disable 명령은 명령을 실행하는 그 때 한번만 유효한 것이 아니라 영구적이다. 명령을 입력하여 실행하게 되면 이클립스에서 실행하거나 새 터미널 세션을 시작한다 해도 프로젝트 전체에 대한 이슈를 더 이상 보고하지 않는다. 그렇기 때문에 임시로 이슈를 제외하고 싶다면 실행하고 난 뒤에는 반드시 다시 복구해야 한다. 이 때는 --enable 인자를 추가하면 되는데, 이렇게 하면 Lint가 다시 실행되고 제외됐던 이슈가 출력된다. 그 출력 결과는 다음과 같다.

```
$ lint ./ToolsDemo --enable Internationalization,ContentDescription

Scanning ToolsDemo: .....
Scanning ToolsDemo (Phase 2): .....
res/layout/gooduidemo.xml:15: Warning: Should use "sp" instead of "dp" for
text sizes [SpUsage]
```

```
android:textSize="20dp" />
```

```
res/layout/baduidemo.xml:167: Warning: [I18N] Hardcoded string "Text will go here", should use @string resource [HardcodedText]
```

```
android:hint="Text will go here"
```

...

0 errors, 80 warnings

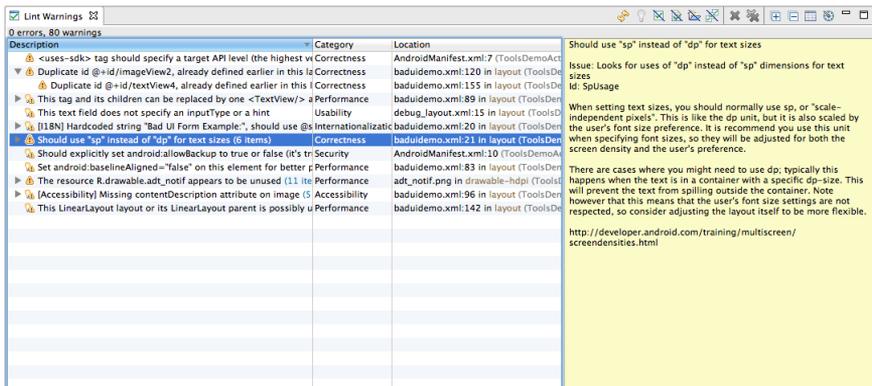
\$

1.3.2 이클립스에서 실행하기

이클립스 안에서도 Lint를 쉽게 실행할 수 있다. 프로젝트 폴더에서 오른쪽 클릭 후 Android Tools → Run Lint: Check for Common Errors를 선택하면 된다.

Lint를 실행하면 그림 1-12와 같이 'Lint Warnings'라는 탭이 새로 생김을 볼 수 있는데, 이 Lint UI에서 오류를 확인하고 해결할 수 있다. 이슈의 종류에 따라 정리된 오류 트리가 있는데, 이 오류 트리로는 특정 이슈 등에 집중할 수 있다. 툴바에는 목록을 조작할 수 있는 다양한 명령과 출력 관련 선택사항이 있다.

그림 1-12 Lint Warning 탭



Lint 툴바 메뉴

그림 1-13과 같이 오류를 강조 표시하면 탭 상단에 위치한 툴바가 활성화된다. 이 'Lint Warnings' 툴바에서 이슈 제외를 한번에 조정할 수 있다.

그림 1-13 Lint Warnings 툴바



The buttons, from left to right, offer the following tasks:

툴바의 버튼(왼쪽에서 오른쪽)은 다음 기능을 수행한다.

Refresh

Lint 검사를 다시 수행하고 새 결과를 표시한다.

Fix

XML 또는 Java 편집기를 시작하고 이슈를 해결하기 위해 소스를 수정한다.

Suppresses the selected error with an annotation/attribute

소스 파일에 표시를 삽입해 특정 인스턴스에서는 경고나 오류를 생성하지 않는다.

Ignore in this file

선택한 이슈는 파일 전체에서 경고나 오류를 발생하지 않는다.

Ignore in this project

선택한 이슈는 프로젝트 전체에서 경고나 오류를 발생하지 않는다.

Always ignore

어떤 프로젝트나 파일에서도 선택한 이슈의 경고나 오류를 발생하지 않는다.

Remove

현재 출력에서 선택된 이슈를 삭제하지만 이슈를 제외하는 것은 아니다. 다음 실행 시에는 다시 나타난다.

Remove all

현재 출력에서 모든 이슈를 삭제하지만 Lint를 다시 실행하면 다시 나타난다.

Expand all

이슈 트리의 모든 노드를 확장해 개별적인 이슈들을 모두 살펴볼 수 있다.

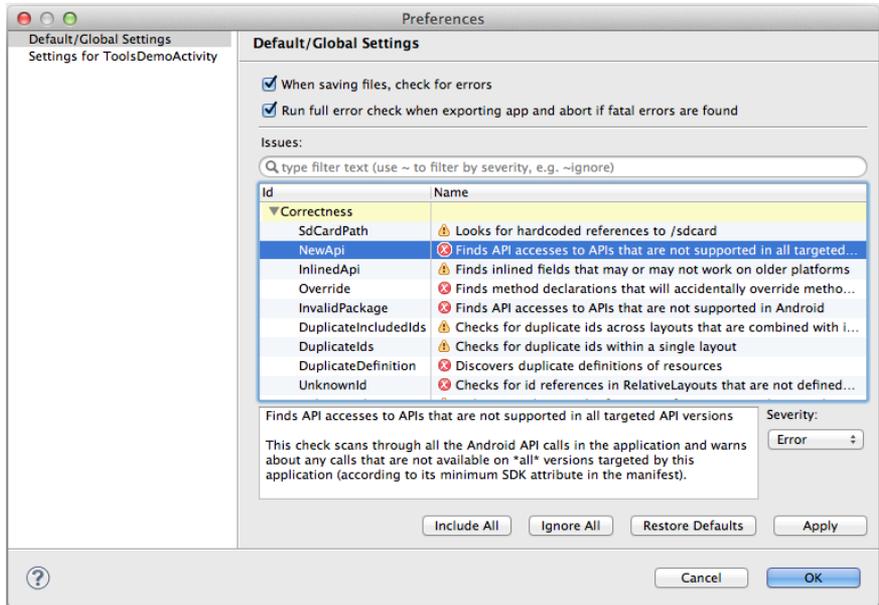
Collapse all

이슈 트리 노드를 축소해 항목을 카테고리별로 묶는다.

Options

그림 1-14처럼 몇 가지 중요한 선택사항을 보여주는 대화창을 표시한다. 이 대화창은 이슈를 포함 또는 제외하는 한 가지 방법을 제공하고, 포함된 이슈들을 확인한다. 여기서 Lint가 언제 어떤 방식으로 실행될지를 설정할 수 있다.

그림 1-14 Lint 선택사항



Java와 XML 편집기 연동

Lint는 기본적으로 자동으로 실행되기 때문에 Lint 오류를 본 적이 있을 것이다. 어떤 선택사항을 지정하느냐에 따라 코드 입력이나 파일 저장 시 오류를 볼 수 있다. Lint 오류가 발생하면 코드의 해당 줄에 표시가 된다. 이슈를 좀 더 자세히 알고 싶으면 경고 아이콘 또는 노란색으로 밑줄 쳐진 코드 부분에 마우스 커서를 갖다 댈다. XML과 Java 편집기에서 어떻게 보이는지 그림 1-15와 1-16을 보자.

그림 1-15 XML 파일 내 Lint 경고

```
13
14 <TextView|
15     android:id="@+id/tool_demo_text"
16     android:layout_width="wrap_content"
17     android:layout_height="wrap_content"
18     android:layout_gravity="center_horizontal"
19     android:layout_marginTop="8dp"
20 [!18N] Hardcoded string "Bad UI Form Example:", should use @string resource
21     android:textSize="20dp" />
22
23 <Button
```

그림 1-16 자바 파일 내 Lint 경고

```
1 package com.tools.demo;
2
3 Multiple markers at this line
4 - The import android.widget.TextView is never used
5 - The import android.content.Intent is never used
6
7
8 private Button eatMemory;
9 private Vector vect;
10
11
12 /** Called when the activity is first created. */
13 @Override
14 public void onCreate(Bundle savedInstanceState) {
15     super.onCreate(savedInstanceState);
16
17 }
```

Quick Fix 도구

코드의 오류 정보만 알면 안 되고 문제를 해결해야 한다. Quick Fix 도구를 사용하면 오류 해결을 무척 쉽게 할 수 있다.

Quick Fix 도구를 사용하는 가장 좋은 방법은 오류가 발생한 코드에서 윈도우 또는 리눅스에서는 Ctrl + 1, Mac OS X에서는 Command + 1을 누르는 것이다.

Quick Fix 대화창에서 문제를 해결할 수 있는 다양한 선택항목을 볼 수 있는데, 이 항목은 Lint 툴바와 같은 선택항목이다. 이 선택항목은 XML과 Java에서 각각 그림 1-17과 1-18과 같이 표시된다.

이 기능이 모든 이슈를 다 해결해 주지는 않는다. 시스템에서 Quick Fix를 제공하지 경우에는 코드를 직접 디버그하여 해결 방법을 찾아야 한다.

그림 1-17 XML에서 Lint Quick Fix

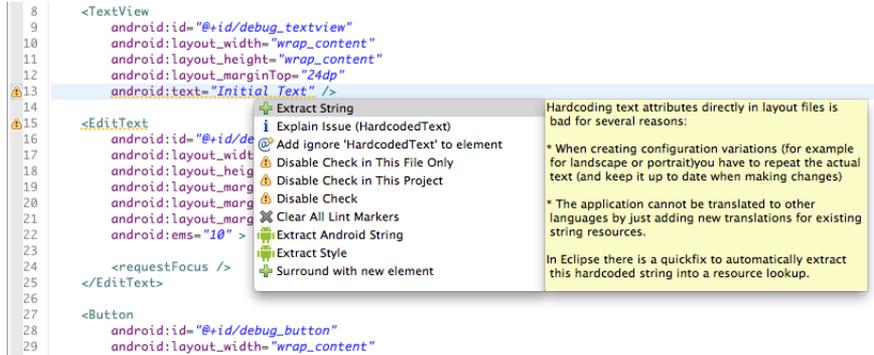
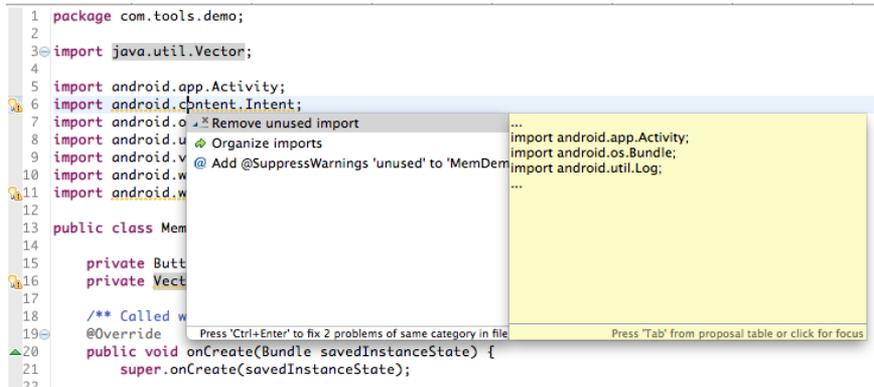


그림 1-18 Java에서 Lint Quick Fix



2 | 이벤트 시뮬레이션

테스트를 해 보고 싶지만 실제 상황처럼 재현하기가 어려운 경우가 많다. 이럴 때는 이벤트 시뮬레이션을 가능하게 해 주는 도구를 사용하면 좀 더 효과적으로 테스트할 수 있다.

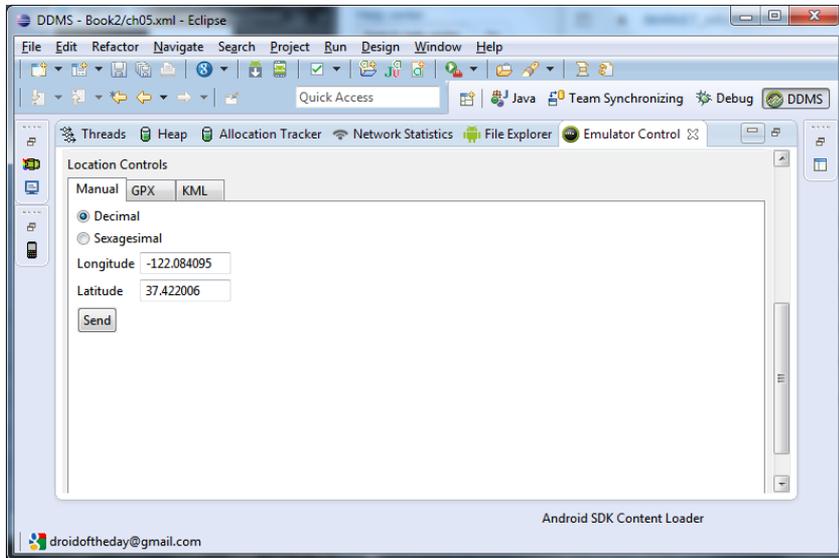
2.1 위치와 경로 시뮬레이션

위치 테스트는 어려울 수 있는데, 여러 곳을 돌아다니거나 같은 구간을 계속해서 돌아다니는 것은 실용적이지 못하거나 불가능할 수 있다. 다행히도, DDMS 도구로 에뮬레이터에서 위치(위도/경도 좌표)를 지정하거나 경로(GPX 또는 KML 형식)를 시뮬레이션할 수 있다.

위치 시뮬레이션을 하려면 DDMS 도구를 실행해야 한다('4.2.1 DDMS Perspective 실행하기' 참조). 우선 Window → Open Perspective... → Other... → DDMS → OK를 선택하여 Eclipse perspective를 실행한다.

다음은 'Emulator Control' 탭을 연다(그림 2-1과 같이 보인다). 이 탭에서 'Location Controls'라고 표시된 섹션을 볼 수 있는데 여기에 위치 속성을 입력한다. 데이터 입력이 끝나면 Send, Load GPX, 또는 Load KML 버튼(작업하는 데이터에 따라 달라질 수 있다)을 클릭하면, 지정한 위치에 대한 시뮬레이션이 시작된다.

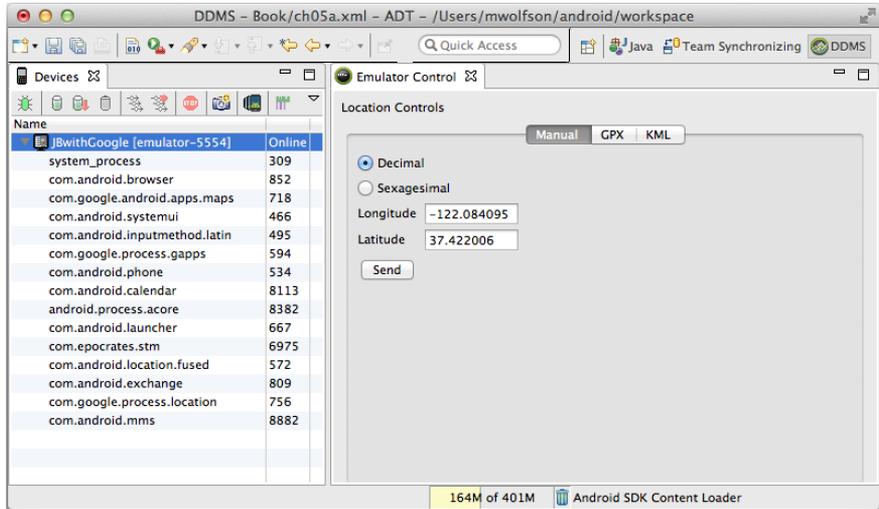
그림 2-1 에뮬레이터로 위치 시뮬레이션하기



기기에서 특정 위치를 시뮬레이션하는 단계는 다음에 나와있다. 이 단계는 KML 또는 GPX를 사용하는 경로 시뮬레이션에서도 동일하다.

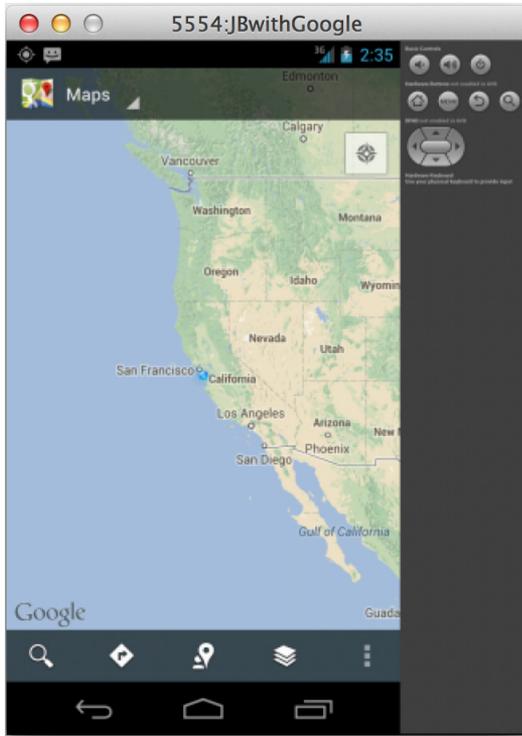
1. 이클립스에서 DDMS perspective 실행한다.
2. 작업할 기기 또는 에뮬레이터를 찾아서 Devices 탭에서 선택한다.
3. 오른쪽 패널에서 Emulator Control 탭을 선택한다(그림 2-2).
4. Telephony Actions 섹션에서, Location Controls라고 되어 있는 아래쪽 섹션까지 스크롤한다. 이 섹션에서 Manual 탭을 선택하고, 유효한 위도와 경도를 양식에 입력한다. 기본적으로 입력된 값은 미국 캘리포니아 주 마운틴 뷰 Mountain View다.

그림 2-2 위치 시뮬레이션 설정



5. 이미 설정되어 있지 않은 경우에는 기기에서 Location Settings를 활성화해야 한다(그림 2-3). 위치 정보에 접근할 때 처음에는 사용자 동의 화면이 뜬다..
6. 기기의 위치를 시뮬레이션하려면 Send 버튼을 누른다. 그러면 기기에 해당 위치가 반영된다.

그림 2-3 위치 시뮬레이션 보기

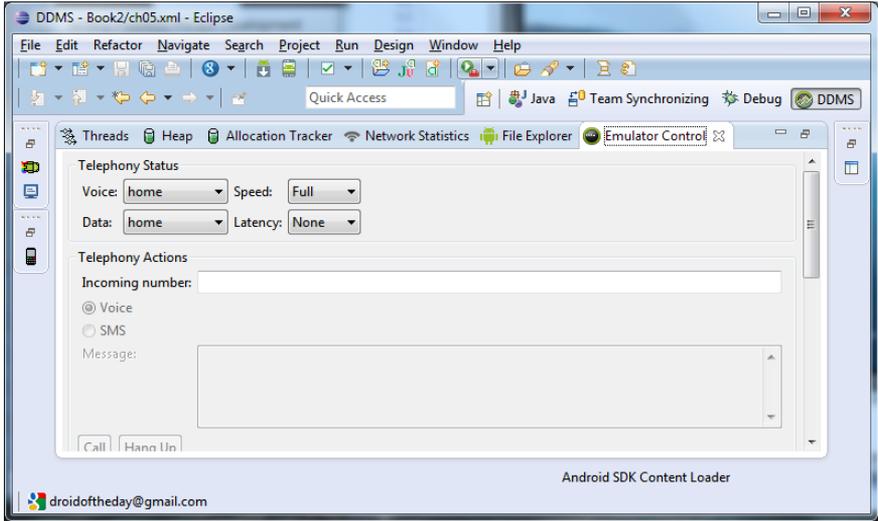


2.2 전화 관련 시뮬레이션

Emulator Control 탭에는 Telephony Actions라는 섹션도 있다(그림 2-4). 여기서 전화 이벤트와 무선 연결 기능을 시뮬레이션할 수 있는데, 전화와 문자 메시지 관련 기능의 시뮬레이션을 에뮬레이터에서 할 수 있다. 무선연결이 불안정할 때도 앱이 제대로 동작하는지를 확인하기 위해 무선 설정을 조정할 때도 유용하다. 탭의 가장 상단에는 Telephony Status 섹션이 있다. 연결 문제를 시뮬레이션하려면(응답이 늦거나 패킷 손실이 발생하는 경우 등) 여기의 설정을 조정하면 된다. 다음 Telephony Actions 섹션은 전화와 문자 메시지 송수신을 테스트해 볼 수 있다.

이 두 가지 기능 중 하나를 시뮬레이션하려면, 회신할 전화번호를 입력하고 원하는 동작(전화 걸기 또는 SMS)을 선택한다. 문자 메시지의 경우라면 메시지 본문을 입력하고 Send 버튼을 클릭한다.

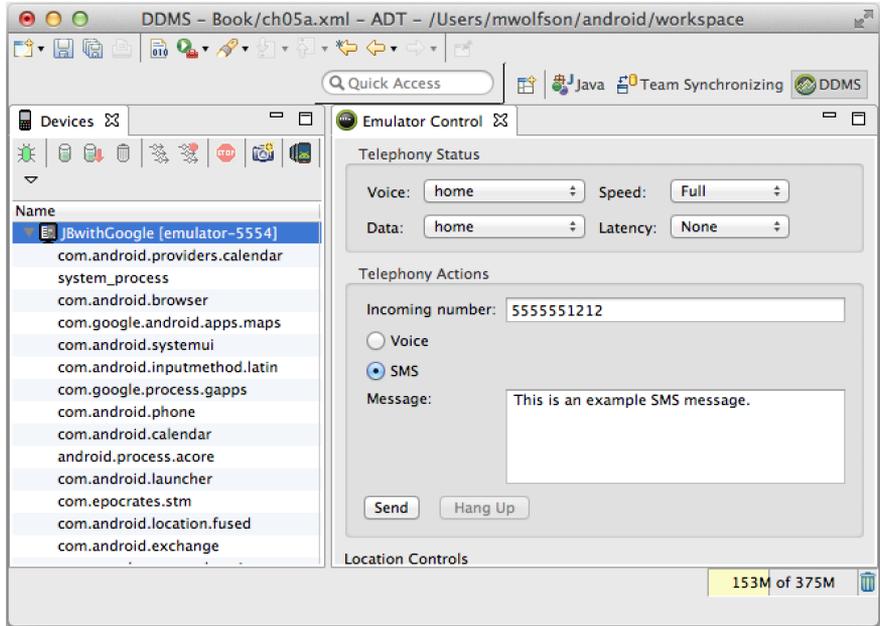
그림 2-4 에뮬레이터의 전화 시뮬레이션



다음은 SMS 메시지를 생성하는 방법이다. 전화 걸기 시뮬레이션도 같은 과정으로 할 수 있다.

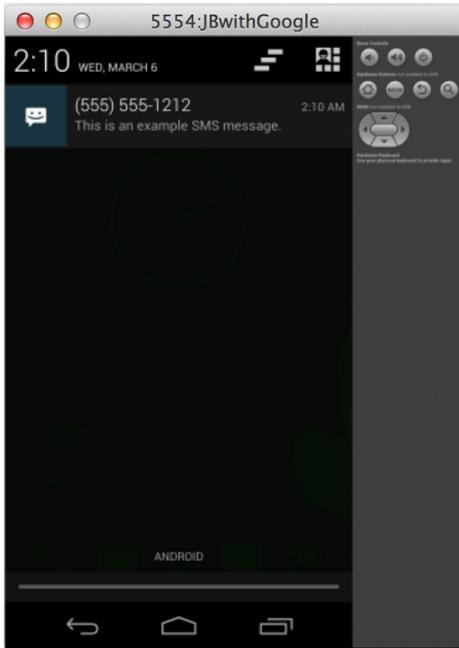
1. 이클립스에서 DDMS perspective를 실행한다.
2. 작업할 기기 또는 에뮬레이터를 찾아서 Devices 탭에서 선택한다.
3. 그림 2-5처럼 오른쪽 창에서 Emulator Control 탭을 선택한다.

그림 2-5 SMS 시뮬레이션 설정



4. (선택사항) 연결 상태가 좋지 않은 상황에서 앱을 테스트하고자 할 때 Speed와 Latency 설정을 변경하면 된다.
5. Telephony Actions 섹션에서 전화번호(하이픈 없이)와 메시지 텍스트를 입력한다.
6. 전송할 메시지를 보내려면 Send 버튼을 누르면 되는데, 메시지는 기기 또는 에뮬레이터에서 표시된다. 그림 2-6처럼 기기는 시스템에 보내진 SMS를 표시한다.

그림 2-6 SMS 시뮬레이션 보기



2.3 네트워크 파라미터 변경

기기의 네트워크 파라미터를 변경할 때가 있는데, 이는 컴퓨터에서 에뮬레이터나 기기로 요청을 전달할 때 유용하다(안드로이드 시스템의 로컬 네트워크 설정을 테스트해야 할 때 등 있다).

ADB를 사용하면 이를 쉽게 할 수 있으며, 문법은 다음과 같다.

```
adb forward tcp:9222 tcp:9333
```

그런 다음 로컬 데스크톱에서 ping을 보내면 명령이 바로 안드로이드 기기로 전

달된다.

localhost:9333

2.4 센서 에뮬레이션을 기기에 적용하기

멀티 터치 또는 자이로스코프와 같은 동작 기반 센서는 에뮬레이터에서 테스트하기 힘들다. 이 어려움을 극복하기 위해 ADT는 실제 기기를 에뮬레이터에 연결해 기기의 센서가 에뮬레이터와 연동될 수 있도록 해 준다. 에뮬레이터에서 동작하는 앱은 기기의 센서 변화를 감지하고, 이 정보는 에뮬레이터에 전달되어 시스템 이미지로 보내진다. 이 방식으로 각종 센서 이벤트를 실제 기기에서 생성해 동작 중인 에뮬레이터로 보낼 수 있다.

이 기능을 사용하려면, 안드로이드 4.0 release 2 이상의 시스템 이미지가 에뮬레이터에서 구동 중이어야 한다.

센서 에뮬레이션을 활성화하려면 다음 단계를 따른다.

1. 사용하는 AVD를 수정한다. 하드웨어 속성에 'Multi-touch screen support'를 추가하고 'true'로 설정한다.
2. SdkControllerSensor 앱을 장치에 설치한다. \$SDK/tools/apps/SdkController 폴더에서 앱의 소스 코드가 있다.
3. 기기의 'USB debugging'을 활성화하고 컴퓨터에 연결한다.
4. SdkControllerSensor 앱을 기기에서 실행한다.
5. 앱에서 에뮬레이트할 센서를 선택한다.

6. 다음 명령을 기기의 셸에서 실행해 포트 포워딩을 활성화한다.

```
adb forward tcp:1968 tcp:2068
```

7. 테스트할 에뮬레이터를 실행한다.

포트 포워딩은 불안정할 수 있다. 에뮬레이터에서 센서 이벤트가 감지되지 않으면 'adb forward tcp:1968 tcp:2068' 명령을 다시 실행해 연결을 복구한다.

2.5 센서 고급 테스트

센서 활용이 많은 앱을 제작하고 있다면 센서를 테스트하는 데 상당히 많은 문제를 접할 것이다. 상황마다 테스트하는 것은 힘들며 많은 경우 불가능할 때도 있다. 또한, 극한의 상황을 테스트하는 것도 현실적으로 어려울 수 있다. 예를 들어, 극단적인 온도를 테스트하려고 전화기를 오븐이나 냉동실에 넣을 수는 없다. 센서의 경우, 정확한 값을 없는 것도 힘들 수 있다(자이로스코프를 테스트하려고 전화기를 오랫동안 가만히 들고 있어볼 수 있을까). 이때, OpenIntents.org에서 관리하는 오픈 소스 프로젝트 [SensorSimulator](https://github.com/OpenIntents/SensorSimulator)⁰¹를 이용하면 센서 테스트를 훨씬 수월하게 할 수 있다.

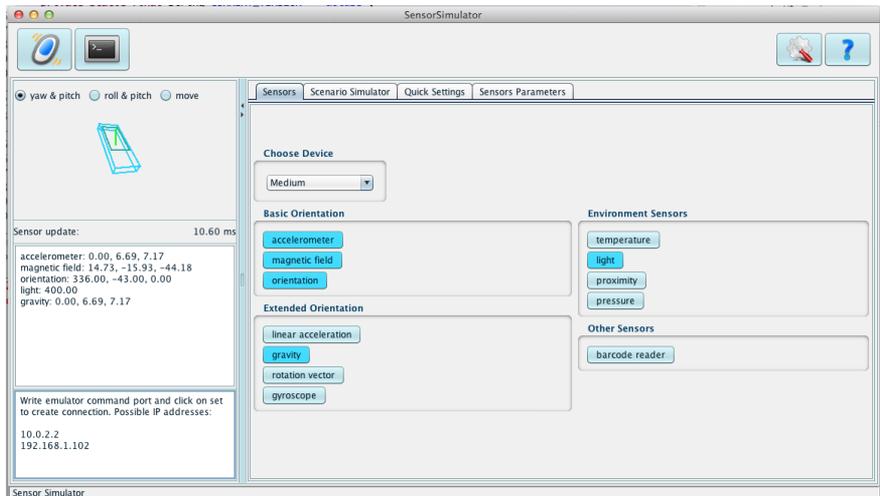
SensorSimulator는 데스크톱 구성요소와 여러 APK를 포함한 일련의 앱인데, 데스크톱 구성요소를 사용해 실시간 센서 이벤트를 기기에 보낼 수 있다. 센서를 미세한 수준까지 조작할 수 있고 원할 때 이벤트를 발생시킬 수 있다는 점은 센서를 사용하는 앱을 제작할 때 아주 유용하다. 이 도구는 일련의 센서 이벤트를 기록해 기기에서 재현할 수 있게 해 주며, 시나리오를 생성해(데스크톱 앱을 사용하거나 기기에서 이벤트를 기록하는 방식으로) 저장한 다음 다시 재생할 수 있다. 같은 센서 동작을 몇 번이고 반복할 수 있게 해 주기 때문에 앱 동작 테스트가 매우 편리하다.

01 <http://bit.ly/16zLVem>

2.5.1 지원되는 센서

SensorSimulator 프로젝트는 가속센서, 나침반, 방향, 온도, 조명, 근접, 압력, 중력, 선형 가속, 회전각, 자이로스코프 센서를 포함한 여러 센서를 지원한다. Sensors 탭에서 각 센서의 시뮬레이션을 설정할 수 있다. 센서는 이 창에서 활성화 또는 비활성화할 수 있는데(그림 2-7의 오른쪽), 테스트하고자 하는 센서만을 활성화해 관심 있는 특정 데이터만을 볼 수도 있다. 센서의 값을 조정하고 싶을 때는 Sensors Parameters 탭의 Quick Settings에서 하면 된다.

그림 2-7 SensorSimulator에서 지원하는 센서



2.5.2 실시간 센서 이벤트 시뮬레이션

이 앱의 내려받기, 설치, 실행은 일반적인 앱과 다를 바 없다(프로젝트 웹 사이트에 간단한 설명이 있다). 데스크톱 자바 앱(bin/SensorSimulator.jar)을 컴퓨터에서 실행하고 기기에 APK를 설치하여 실행한다(bin/SensorSimulatorSettings-x.x.x.apk). WiFi 연결을 통해 두 프로세스를 연결한다. 프로세스가 연결되면 데스크톱 앱을 이용해 센