Analisi statica

Principali tecniche di analisi statica

1. Analisi statica in compilazione
2. Code reading
3. Code inspections or reviews
4. Walktrough
5. Control flow analysis
6. Data flow analysis
7. Esecuzione simbolica

1- Analisi statica in compilazione

- I compilatori effettuano una analisi statica del codice per verificare che un programma soddisfi particolari caratteristiche di correttezza statica, per poter generare il codice oggetto.

- Tipiche anomalie identificabili:
  - *nomi di identificatori non dichiarati,*
  - *incoerenza tra tipi di dati coinvolti in una istruzione,*
  - *incoerenza tra parametri formali ed effettivi in chiamate a subroutine,*
  - *codice non raggiungibile dal flusso di controllo*

2 - Code Reading (o Desk Checking)

- E' effettuata un'attenta lettura individuale del codice per individuare errori e/o discrepanze con il progetto.
- Il lettore effettua mentalmente una pseudo-esecuzione del codice e processi di astrazione che lo conducono a verificare la correttezza del codice rispetto alle specifiche e il rispetto di standard adottati.
- Tipici difetti identificabili:
  - nomi di identificatori errati, errato innesto di strutture di controllo, loop infiniti, inversione di predicati, commenti non consistenti con il codice, incorretto accesso ad array o altre strutture dati, incoerenza tra tipi di dati coinvolti in una istruzione, incoerenza tra parametri formali ed effettivi in chiamate a subroutine, inefficienza dell'algoritmo, non strutturazione del codice, codice morto, etc.
- L'efficacia di tale tecnica è limitata se chi la esegue è la stessa persona che ha scritto il codice.

3 - Code Inspections (o review)

- Riunioni formali cui partecipa un gruppo di persone tra cui almeno una del gruppo di sviluppo, un moderatore ed altri esperti.
- Lo sviluppatore legge il codice ad alta voce, linea per linea, e i partecipanti fanno commenti e/o annotazioni.
- Tipicamente queste riunioni sono preannunciate ai partecipanti cui viene fornita la documentazione necessaria (codice e relativi documenti) per la revisione.
- È una tecnica che riesce ad individuare fra il 30 e il 70% degli errori nella logica del programma.

Code Inspection

- L'obiettivo della riunione è scoprire difetti, non correggerli. Comunque, spesso l'analisi dei difetti effettuata viene discussa e vengono decise le eventuali azioni da intraprendere
  - accettazione del codice, rigetto, annotazioni su eventuali non aderenze a specifiche, indicazioni delle modifiche da apportare

- Il codice è analizzato usando **checklist** dei tipici errori di programmazione, quali:
  - Errori di data reference, data declaration, di calcolo, di confronto, sul flusso di controllo, di interfaccia, di I/O.
  - Le checklist sono generiche (indipendenti dal linguaggio) ma possono essere adattate agli specifici linguaggi analizzati.

Esempio di Checklist per le Ispezioni

Inspection Error Checklist Summary, Part I

Data Reference	*Computation*
1. Unset variable used?	1. Computations on nonarithmetic variables?
2. Subscripts within bounds?	2. Mixed-mode computations?
3. Non integer subscripts?	3. Computations on variables of different lengths?
4. Dangling references?	4. Target size less than size of assigned value?
5. Correct attributes when aliasing?	5. Intermediate result overflow or underflow?
6. Record and structure attributes match?	6. Division by zero?
7. Computing addresses of bit strings? Passing bit-string arguments?	7. Base-2 inaccuracies?
8. Based storage attributes correct?	8. Variable's value outside of meaningful range?
9. Structure definitions match across procedures?	9. Operator precedence understood?
10. Off-by-one errors in indexing or subscripting operations?	10. Integer divisions correct?
11. Are inheritance requirements met?	

Data Declaration
1. All variables declared?
2. Default attributes understood?
3. Arrays and strings initialized properly?
4. Correct lengths, types, and storage classes assigned?
5. Initialization consistent with storage class?
6. Any variables with similar names?

Comparison
1. Comparisons between inconsistent variables?
2. Mixed-mode comparisons?
3. Comparison relationships correct?
4. Boolean expressions correct?
5. Comparison and Boolean expressions mixed?
6. Comparisons of base-2 fractional values?
7. Operator precedence understood?
8. Compiler evaluation of Boolean expressions understood?

Alcuni Strumenti di analisi del codice in ambiente java

- **Tra gli strumenti di analisi statica più noti, in ambiente Java:**
  - PMD
  - Checkstyle
  - Findbugs
- **In ambiente Android**
  - Lint

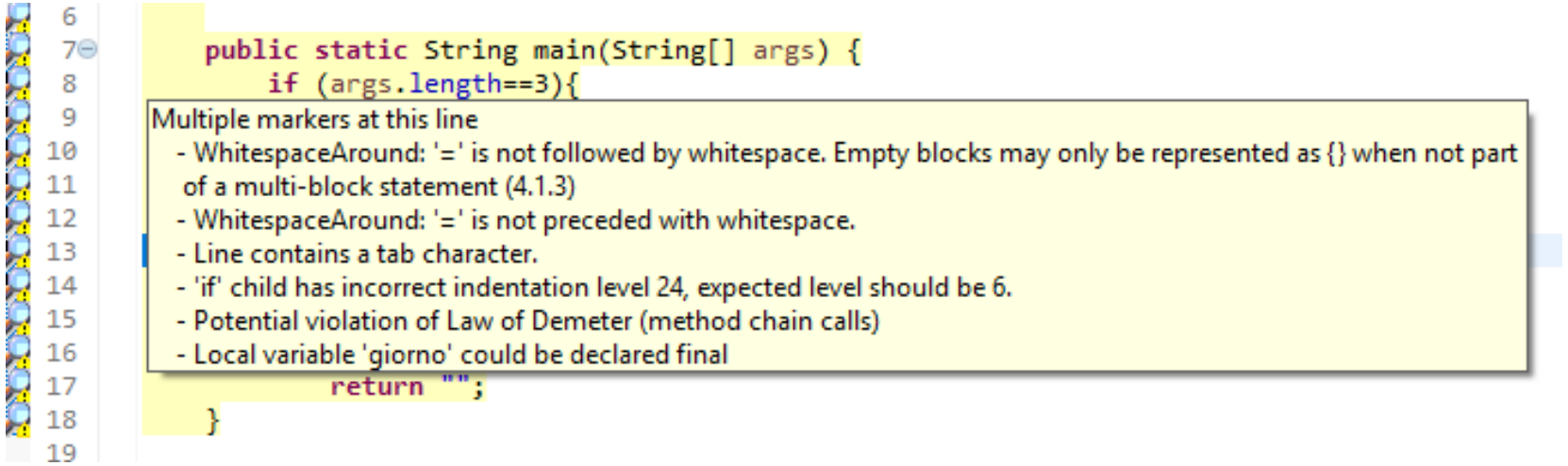
- **Tutti questi strumenti sono disponibili sia standalone, spesso integrati in ant, maven o gradle, che come estensioni e plugin degli ambienti di sviluppo**

Strumenti per l'analisi statica : checkstyle

- Checkstyle
  - http://checkstyle.sourceforge.net/
    - *Checkstyle concentra le sue analisi su regole stilistiche e di formattazione*
    - *La pagina ufficiale riporta i link a molti plug in esistenti*
      - Ad esempio in ambiente IntelliJIdea/Android Studio c'è CheckstyleIdea
        - » https://plugins.jetbrains.com/plugin/1065-checkstyle-idea
      - **Il plug in per Eclipse è all'indirizzo** http://eclipse-cs.sourceforge.net/update

Esempio

- **Alcune delle violazioni rilevate da Checkstyle si prestano a correzione automatica (ad esempio quelle relative all'indentazione) e possono essere corrette automaticamente da Checkstyle stesso**



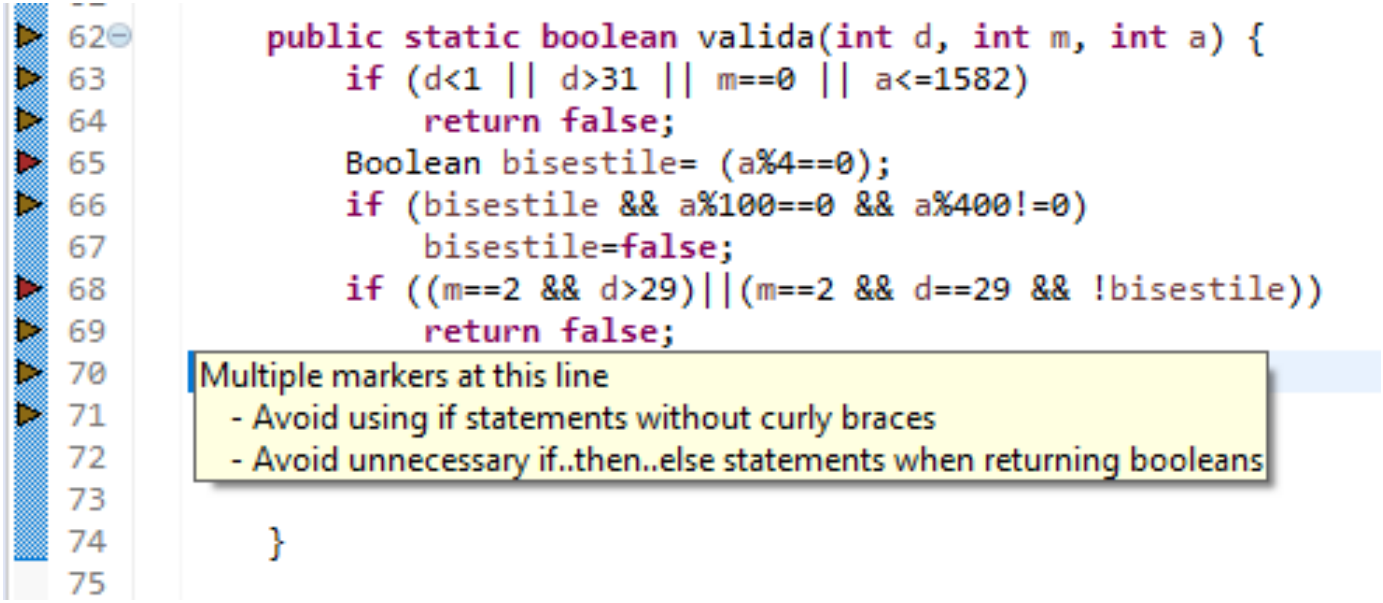
PMD : Programming Mistake Detector

- PMD analizza anch'esso pratiche di cattiva programmazione, ad esempio:
  - Variabili/oggetti non utilizzati
  - Blocchi catch vuoti
  - Codice duplicato
  - Codice troppo complesso (dal punto di vista della complessità ciclomatica)
- La sua forza consiste nel funzionare in una miriade di diversi linguaggi:
  - Java, JavaScript, Apex, Visualforce, PLSQL, Apache Velocity, XML, XSL
  - C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift (solo alcune funzionalità)

- https://pmd.github.io/
- In Eclipse http://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site/
  - Un tutorial per l'uso sotto Eclipse: https://www.javatips.net/blog/pmd-in-eclipse-tutorial

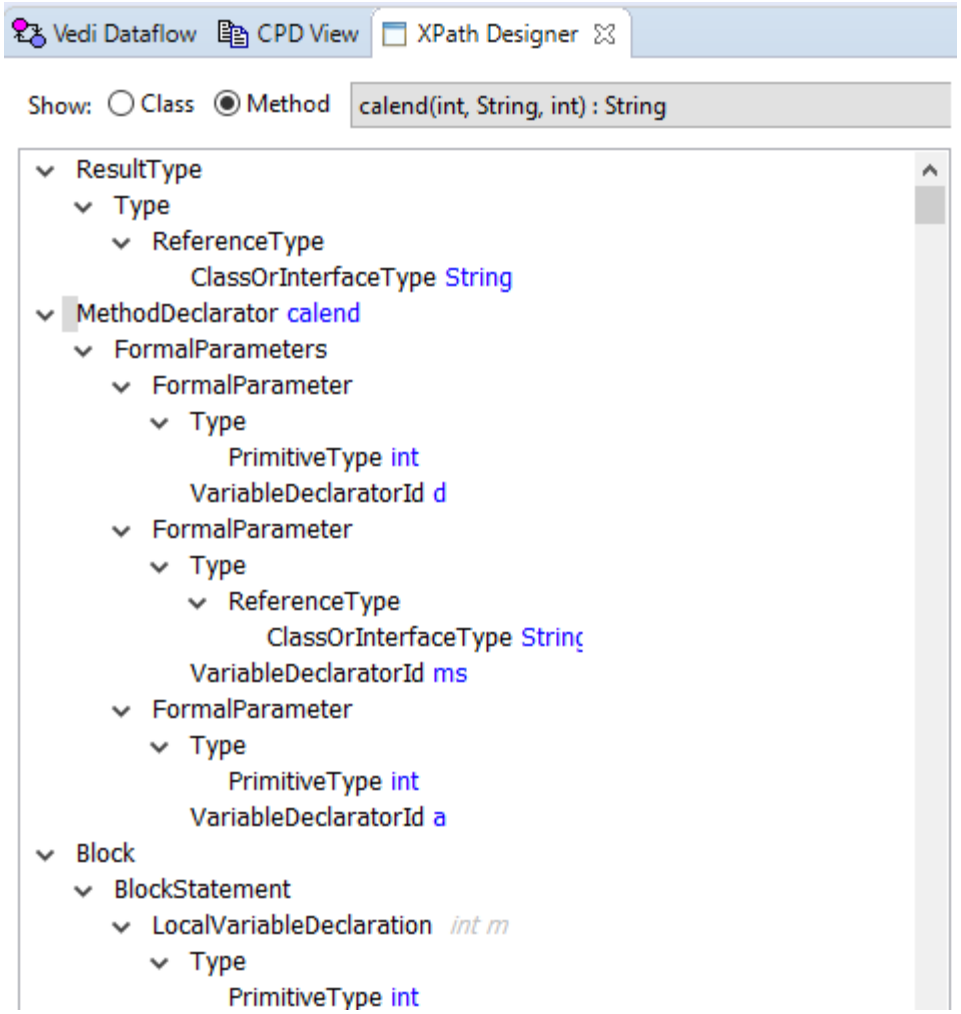
Esempio

- **Per utilizzare PMD è necessario specificare un insieme di regole da applicare. Alcuni insiemi preconfigurati sono all'indirizzo http://pmd.sourceforge.net/pmd-4.3.0/rules/index.html**
- **PMD può integrarsi nello strumento di sviluppo e mostrare le proprie indicazioni già a tempo di sviluppo del codice**



PMD : Abstract Syntax Tree

- **PMD esegue l'analisi statica costruendosi un Abstract Syntax Tree**
- **Tramite quest'albero, PMD fornisce anche ulteriori informazioni di analisi statica, quali CFG e cammini def-use**
- **Inoltre è possibile interrogare tale albero come se fosse un XML (con Xpath) e creare così nuove regole**



Progetto proposto

- **Creare nuove regole per la valutazione di aspetti di qualità del codice, evetualmente tipici di Android**
  - Sfruttando l'Abstract Syntax Tree generato automaticamente da PMD
  - http://pmd.sourceforge.net/pmd-4.3/xpathruletutorial.html
  - Altre risorse: https://pmdapplied.com/
- **Valutare tali regole su progetti di esempio o reali per controllare la loro effettiva frequenza**

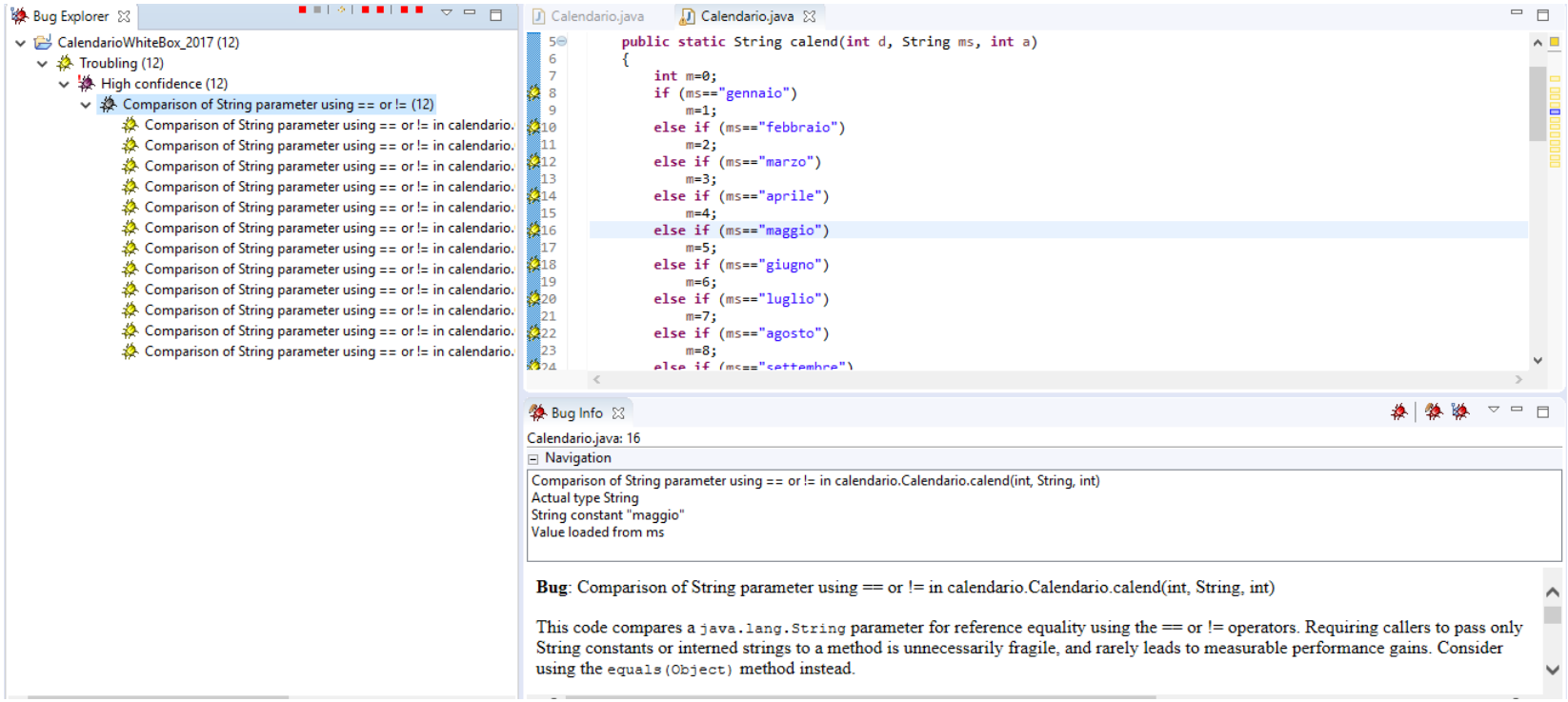
FindBugs

- Findbugs è uno strumento sviluppato dall'Univeristy of Maryland
  - Si concentra soprattutto sulla ricerca di pratiche di programmazione scorrette che potrebbero essere causa di bug
    - *Ovviamente essendo uno strumento di analisi statica non può in generale dimostrare che si tratti di bug non potendo osservare alcun fallimento*

- http://findbugs.sourceforge.net/
- Il plug in per Eclipse può essere installato da: http://findbugs.cs.umd.edu/eclipse
- Tutorial di utilizzo: http://www.vogella.com/tutorials/Findbugs/article.html

Esempio

- **Findbugs fornisce anche una descrizione del problema riscontrato e di come possa essere risolto**



Android Lint

- **Android Lint è uno strumento di analisi statica di applicazioni Android**
  - Eseguibile in maniera standalone
  - Incluso in gradle
  - Eseguibile anche direttamente da Android Studio
    - *E' sufficiente eseguire il task Lint nella scheda Gradle, nel gruppo Verification*
- **Lint cerca potenziali problemi/anomalie come:**
  - Traduzioni mancanti
  - Problemi sul layout
  - Risorse / variabili inutilizzate
  - Dimensioni degli array incoerenti
  - Problemi di internazionalizzazione
    - *Ad es. Stringhe costanti nel codice,*
  - Problemi di accessibilità
    - *Ad es. elementi grafici senza descrizione testuale*
  - Problemi di usabilità
    - *Ad es. Tipo mancante nei campi di input,*

Android Lint

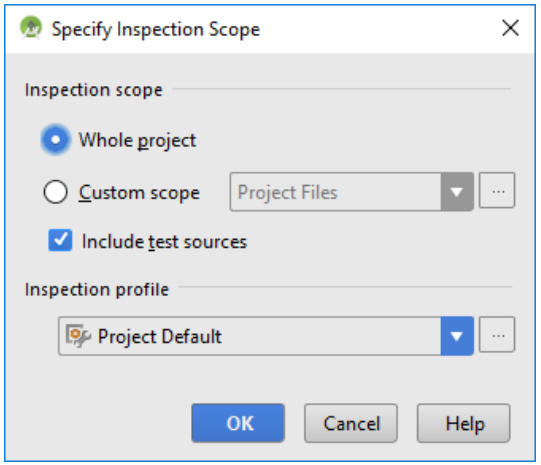
- **Lint cerca potenziali problemi/anomalie come:**
  - Traduzioni mancanti
  - Problemi sul layout
  - Risorse / variabili inutilizzate
  - Dimensioni degli array incoerenti
  - Problemi di internazionalizzazione
    - *Ad es. Stringhe costanti nel codice,*
  - Problemi di accessibilità
    - *Ad es. elementi grafici senza descrizione testuale*
  - Problemi di usabilità
    - *Ad es. Tipo mancante nei campi di input,*
  - Pratiche di programmazione che riducono le performance o aumentano il consumo energetico
    - *Ad es. Variabili con scope troppo ampio*
- **… (è possibile creare ulteriori controlli)**

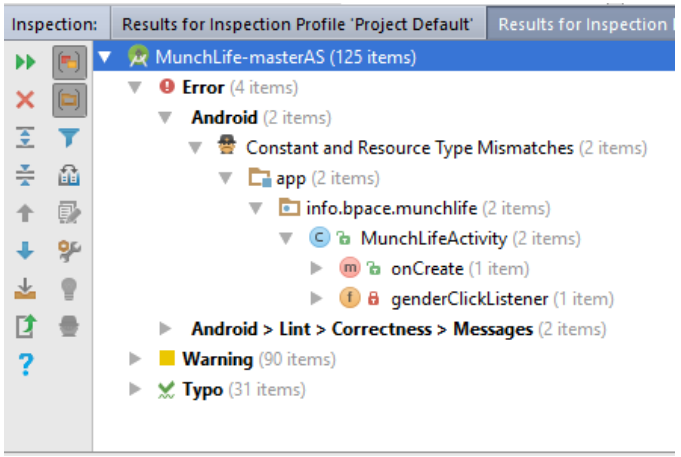
- **Elenco completo dei controlli implementati da Android Lint:**
  - http://tools.android.com/tips/lint-checks

Lint

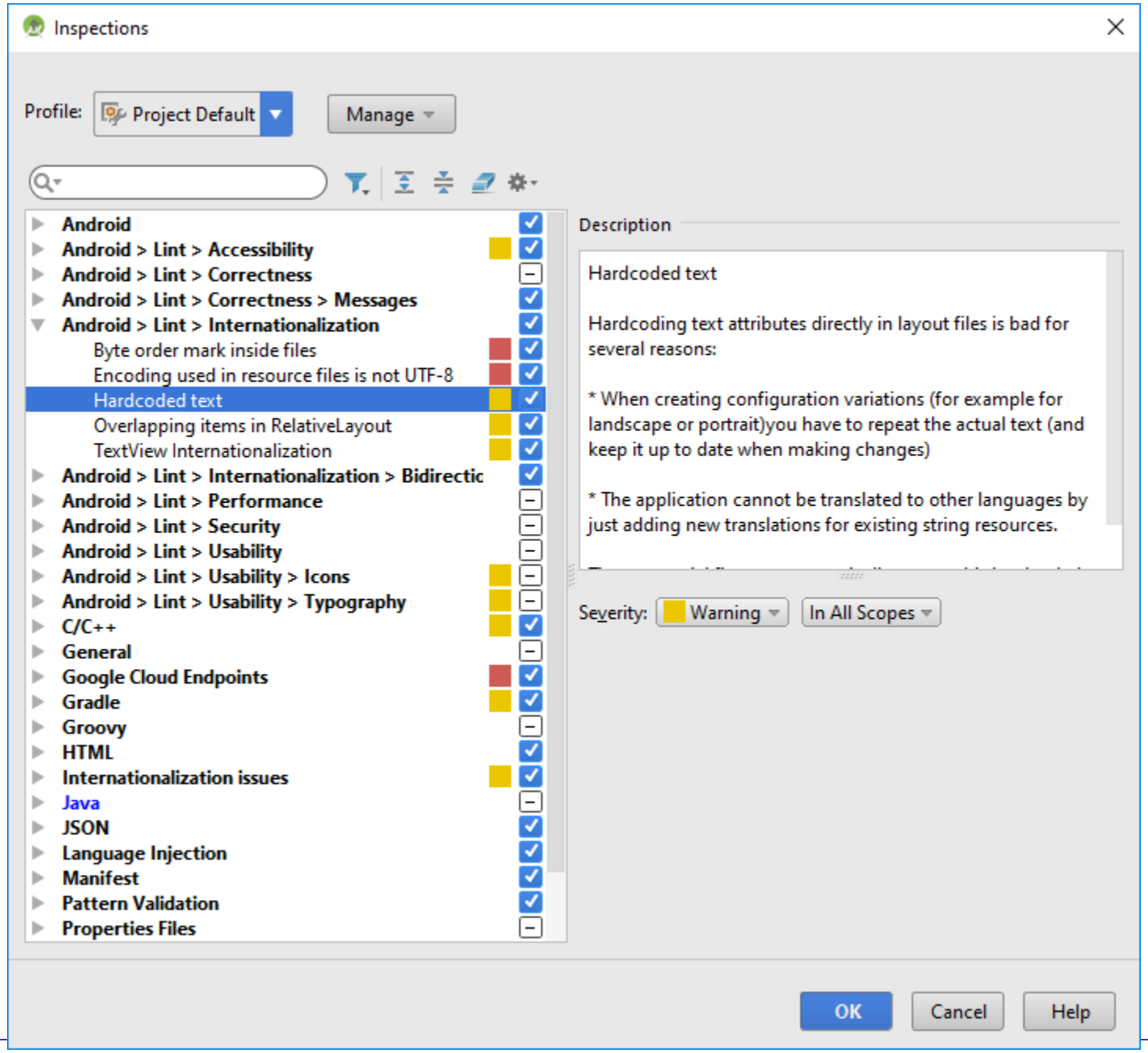
Lint può essere eseguito da Android Studio tramite Analyze/Inspect Code …

Oltre ai controlli di Lint vengono eseguiti alcuni altri controlli di base sul codice sorgente, simili a quelli di PMD, Findbugs, Checkstyle

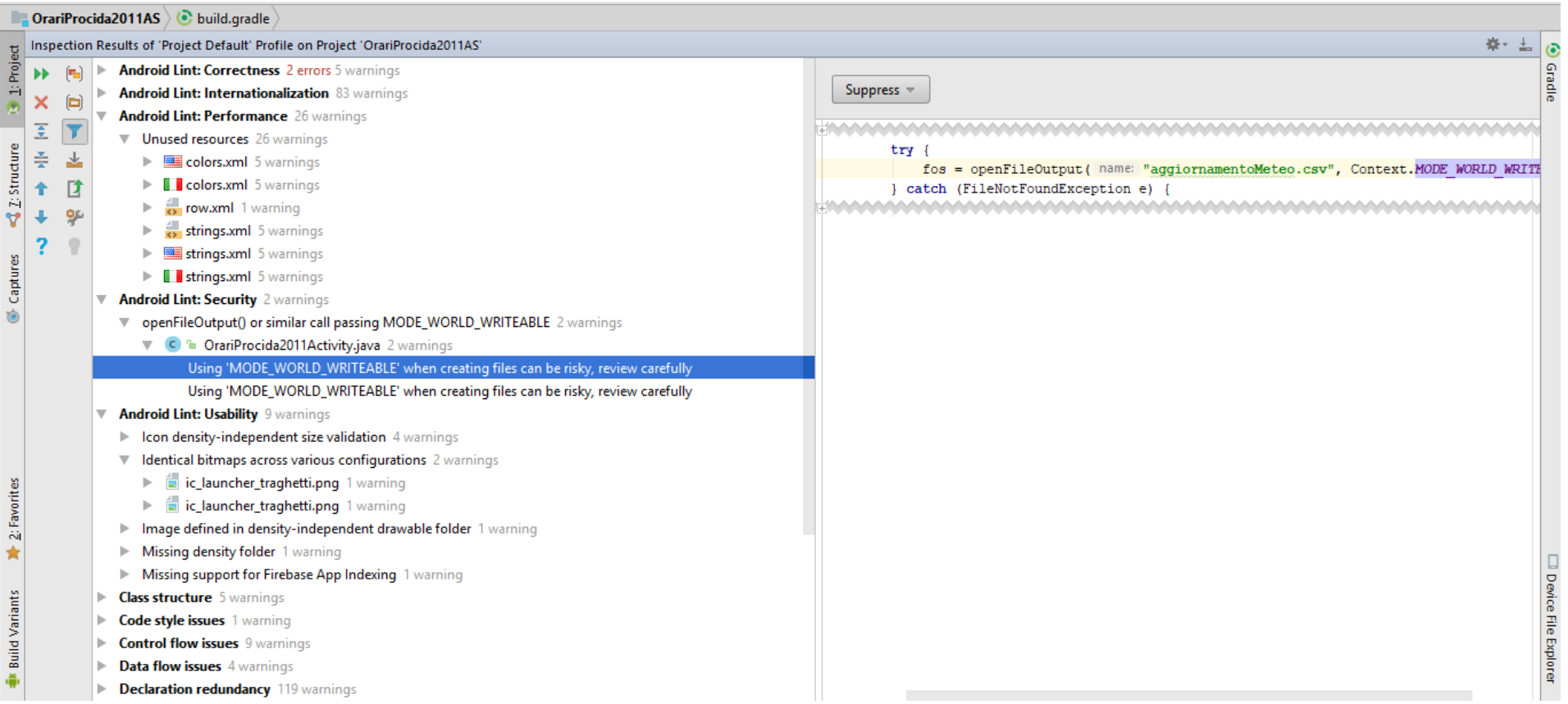




https://developer.android.com/studio/write/lint.html


Inspections
Profile:
Project Default
Manage
Android
Android > Lint > Accessibility
Android > Lint > Correctness
Android > Lint > Correctness > Messages
Android > Lint > Internationalization
Byte order mark inside files
Encoding used in resource files is not UTF-8
Hardcoded text
Overlapping items in RelativeLayout
TextView Internationalization
Android > Lint > Internationalization > Bidirectic
Android > Lint > Performance
Android > Lint > Security
Android > Lint > Usability
Android > Lint > Usability > Icons
Android > Lint > Usability > Typography
C/C++
General
Google Cloud Endpoints
Gradle
Groovy
HTML
Internationalization issues
Java
JSON
Language Injection
Manifest
Pattern Validation
Properties Files
Description
Hardcoded text
Hardcoding text attributes directly in layout files is bad for several reasons:
* When creating configuration variations (for example for landscape or portrait)you have to repeat the actual text (and keep it up to date when making changes)
* The application cannot be translated to other languages by just adding new translations for existing string resources.
Severity:
Warning
In All Scopes
OK
Cancel
Help

Esempio


OrariProcida2011AS
build.gradle
Inspection Results of 'Project Default' Profile on Project 'OrariProcida2011AS'
Android Lint: Correctness 2 errors 5 warnings
Android Lint: Internationalization 83 warnings
Android Lint: Performance 26 warnings
Unused resources 26 warnings
colors.xml 5 warnings
colors.xml 5 warnings
row.xml 1 warning
strings.xml 5 warnings
strings.xml 5 warnings
strings.xml 5 warnings
Android Lint: Security 2 warnings
openFileOutput() or similar call passing MODE_WORLD_WRITEABLE 2 warnings
OrariProcida2011Activity.java 2 warnings
Using 'MODE_WORLD_WRITEABLE' when creating files can be risky, review carefully
Using 'MODE_WORLD_WRITEABLE' when creating files can be risky, review carefully
Android Lint: Usability 9 warnings
Icon density-independent size validation 4 warnings
Identical bitmaps across various configurations 2 warnings
ic_launcher_traghetti.png 1 warning
ic_launcher_traghetti.png 1 warning
Image defined in density-independent drawable folder 1 warning
Missing density folder 1 warning
Missing support for Firebase App Indexing 1 warning
Class structure 5 warnings
Code style issues 1 warning
Control flow issues 9 warnings
Data flow issues 4 warnings
Declaration redundancy 119 warnings
Suppress
try {
fos = openFileOutput(name: "aggiornamentoMeteo.csv", Context.MODE_WORLD_WRITE
} catch (FileNotFoundException e) {
1: Project
7: Structure
Captures
2: Favorites
Build Variants
Gradle
Device File Explorer

Analisi applicazioni Android con Lint

- **Un progetto d'esame precedente ha studiato i risultati relativi all'analisi di circa 800 applicazioni Android con Lint è riportato tra i progetti d'esame svolti**
  - AndroidLInt_DiPaloCarotenutoBuonocore.pdf

Progetto proposto

- **Approfondire l'analisi dei risultati di Lint su applicazioni esistenti per scoprire trend e problemi particolarmente sentiti nell'attuale sviluppo Android**

Oppure

- **Studiare l'opportunità di creare ulteriori regole Lint applicabili ad Android**
  - In particolare, ci sono poche regole che analizzano il codice dei test Android

4 - Walkthrough

- Analisi informale del codice svolta da vari partecipanti i quali 'operano come il computer': in pratica, si scelgono alcuni casi di test e si simula l'esecuzione del codice a mano (si attraversa- walkthrough- il codice).
- L'organizzazione della riunione è simile a quella della tecnica delle Ispezioni:
  - Tra 3 e 5 partecipanti;
  - Riunioni brevi (max. 120 minuti);
  - Attenzione sulla ricerca dei difetti, piuttosto che sulla correzione;
  - Attenzione a non criminalizzare il programmatore (autore del difetto)!

5 - Control Flow Analysis

- Il flusso di controllo è esaminato per verificarne la correttezza.
- Il codice è rappresentato tramite un grafo, il grafo del flusso di controllo (Control flow Graph - CfG), i cui nodi rappresentano statement (istruzioni eo predicati) del programma e gli archi il passaggio del flusso di controllo.
- Il grafo è esaminato per identificare ramificazioni del flusso di controllo e verificare l'esistenza di eventuali anomalie quali codice irraggiungibile e non strutturazione.

6 - Data flow analysis - statica

- Analisi dell'evoluzione del valore delle variabili durante l'esecuzione di un programma, permettendo di rilevare anomalie.
- Intrinsecamente è dinamica, ma alcuni aspetti possono essere analizzati staticamente. L'analisi statica è legata alle operazioni eseguite su una variabile:
  - ***definizione****: alla variabile è assegnato un valore*
  - ***uso****: il valore della variabile è usato in un'espressione oun predicato*
  - ***annullamento****: al termine di un'istruzione il valore associato alla variabile non è più significativo*

- Es.: nell'espressione

  a:=b+c;

  la variabile a è ***definita*** *mentre b e c sono* ***usate***

Data Flow Analysis

- La definizione **(d)** di una variabile, così come un annullamento **(a),** cancella l'effetto di una precedente definizione della stessa variabile, ovvero ad essa è associato il nuovo valore derivante dalla nuova definizione (o il valore nullo)

- Una **corretta sequenza di operazioni** prevede che:
  - L'uso **(u)** di una variabile deve essere sempre preceduto da una definizione della stessa variabile senza annullamenti intermedi (Sequenza valida : du)
    - *Una variabile non definita ha un valore* "sporco"

  - Una definizione di una variabile deve essere sempre seguita da un uso della variabile, prima di un'altra definizione o di un annullamento della stessa variabile (Sequenze non valide: dd, da)
    - *Una doppia definizione è indice del fatto che la prima definizione è risultata inutile*

Data Flow Analysis

Sequenze di istruzioni sono riconducibili a sequenze di *definizioni* (**d**), *usi* (**u**), *annullamenti* (**a**) delle variabili referenziate nei comandi

```
Procedure swap (x1, x2: real)
var x:real
begin
   x2:=x;
   x2:=x1;
   x1:=x;
end;
```

x: (auu)

x2: (ddd)

x1: (dud)

La sequenza (auu) di x e la sequenza (ddd) di x2 sono indicative di una qualche anomalia

---

```
Procedure swap (x1, x2: real)
var x:real
begin
   x:=x2;
   x2:=x1;
   x1:=x;
end;
```

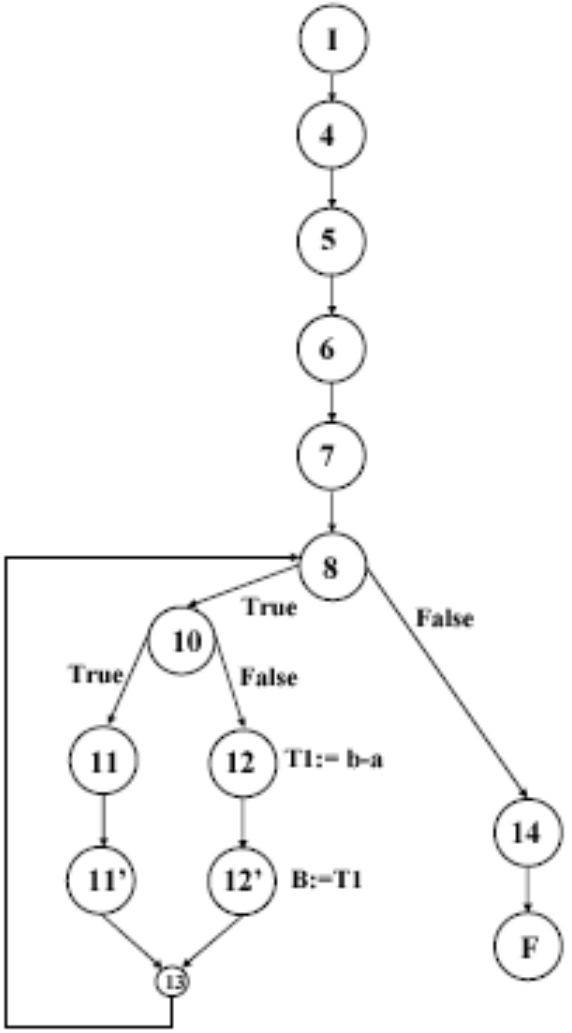
x: (adu)

x2: (dud)

x1: (dud)

Data Flow Analysis-esempio

```
1 program mcd (input, output);
2 var
3 x,y,a,b : integer;
4 begin
5 read (x,y);
6 a:=x;
7 a:=y
8 while a <> b do
9     begin
10        if a>b
11                then a:=a-b
12                else b:=b-a;
13   end;
14 write ("il massimo comune divisore è", a)
15 end.
```



Data Flow Analysis- esempio

A ciascun nodo del CfG è possibile associare l'insieme delle variabili definite in esso, quello delle variabili usate e quello delle variabile annullate.

Nodo	Var. definite	Var usate	Var. annullate
4			x, y, a, b
5	x, y		
6	a	x	
7	a	y	
8		a, b	
10		a, b	
11		a, b	
11'	a		
12		a, b	
12'	b		
14		a	

E' quindi possibile scrivere l'espressione P(*p*;*x*) facendo riferimento a tali insiemi

P([I,4,5,6,7,8,10,11,11',8,14,F];*a*) = (-a-dduuuduu-)

P([I,4,5,6,7,8,10,11,11',8,14,F];*b*) = (-a---uuu-u--)

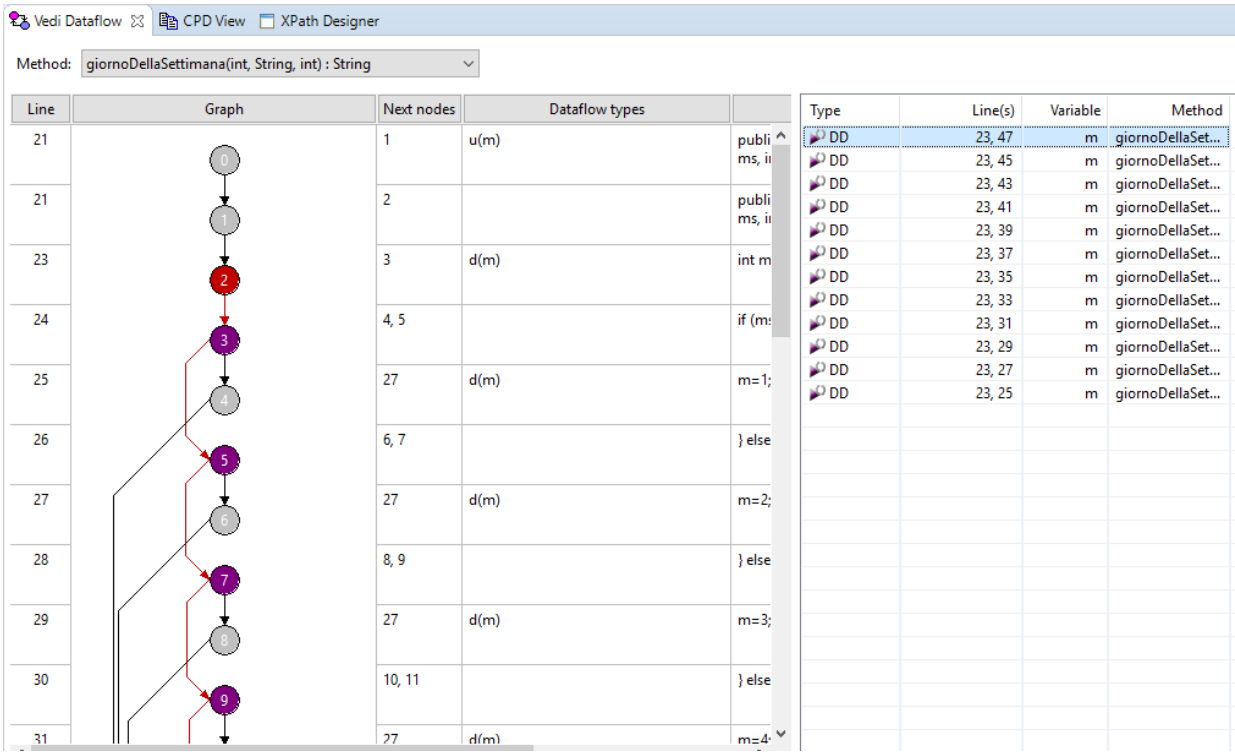
Anomalia ...dd... per *a*
anomalia ...a---u... per *b*
dovuta ad errore a linea 7

```
7  b:= y;
```

Data Flow Analysis con PMD

- PMD è in grado di produrre automaticamente il CFG di un metodo e di tracciare tutte le relazioni tra definizione ed uso delle variabili
  - Disponibili dalla schema «Vedi Dataflow» di PMD

- In quest'esempio si può notare come sia stato rilevato che la variabile m viene prima azzerata (m=0), poi definita in ogni ramo, senza che sia mai utilizzata
  - D'altronde la mancanza di questa prima definizione sarebbe stata segnalata come potenziale problema già dal compilatore



7 - Esecuzione simbolica

- Il programma non è eseguito con i valori effettivi ma con valori simbolici dei dati di input.
- L'esecuzione procede come una esecuzione normale ma non sono elaborati valori, bensì formule formate dai valori simbolici degli input
- Gli output sono formule dei valori simbolici degli input
- L'esecuzione simbolica anche di programmi di modeste dimensioni può risultare molto difficile.
- Ciò è dovuto all'esecuzione delle istruzioni condizionali: deve essere valutato ciascun caso (vero e falso); in programmi con cicli ciò può portare a situazioni difficilmente gestibili.

Esecuzione simbolica

```
1    function product (x,y,z: integer):
integer;
2     var tmp1, tmp2, tmp3: integer;
3     begin
4        tmp1 := x*y;
5        tmp2 := y*z;
6        tmp3 := tmp1 * tmp2 / y;
7     end;
```

Stm.	x	y	z	tmp1	tmp2	product
1	X	Y	Z	?	?	?
4	X	Y	Z	X*Y	?	?
5	X	Y	Z	X*Y	Y*Z	?
6	X	Y	Z	X*Y	Y*Z	(X*Y)*(Y*Z)/Y

Java Path Finder

- Java Path Finder fornisce funzioni per l'esecuzione simbolica di metodi di applicazioni Java
  - In allegato nel materiale docenti c'è un progetto d'esame svolto, che utilizza Java Path Finder per esecuzione simbolica
    - *JavaPathFinder_Cascella.zip*

Path Conditions

- Nel caso di esecuzioni simboliche con condizioni, alcuni statement sono eseguiti solo se gli input soddisfano determinate condizioni.
- Una **Path Condition** *(pc),* **per** un determinato **statement**, indica le condizioni che gli input devono soddisfare affinchè una esecuzione percorra un cammino lungo cui lo statement sia eseguito.
- Una *pc* è un'espressione Booleana sugli input simbolici di un programma.

Path Condition

All'inizio dell'esecuzione simbolica essa assume il valore vero

- (pc := true).

Per ogni condizione che si incontrerà lungo l'esecuzione , pc assumerà differenti valori a seconda dei differenti casi relativi ai diversi cammini dell'esecuzione.

Es.

……

if C then S1 else S2;

……

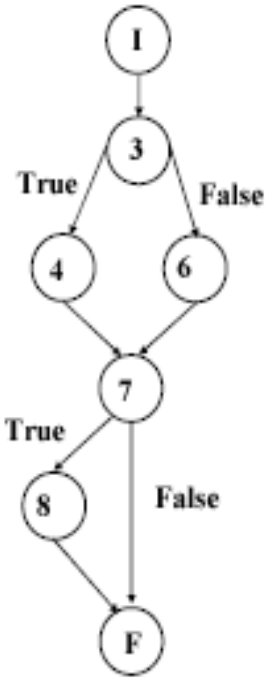
$pc := true$

$pc := pc \ \wedge \ C$ [pc per S1]

$pc := pc \ \wedge \ (not\ C)$ [pc per S2]

```
1   function max (x,y,z: integer): integer;
2   begin
3      if x <= y then
4                max := y
5             else
6                 max := x;
7      if max <= z  then
8                max := z;
9    end;
```

Path Condition



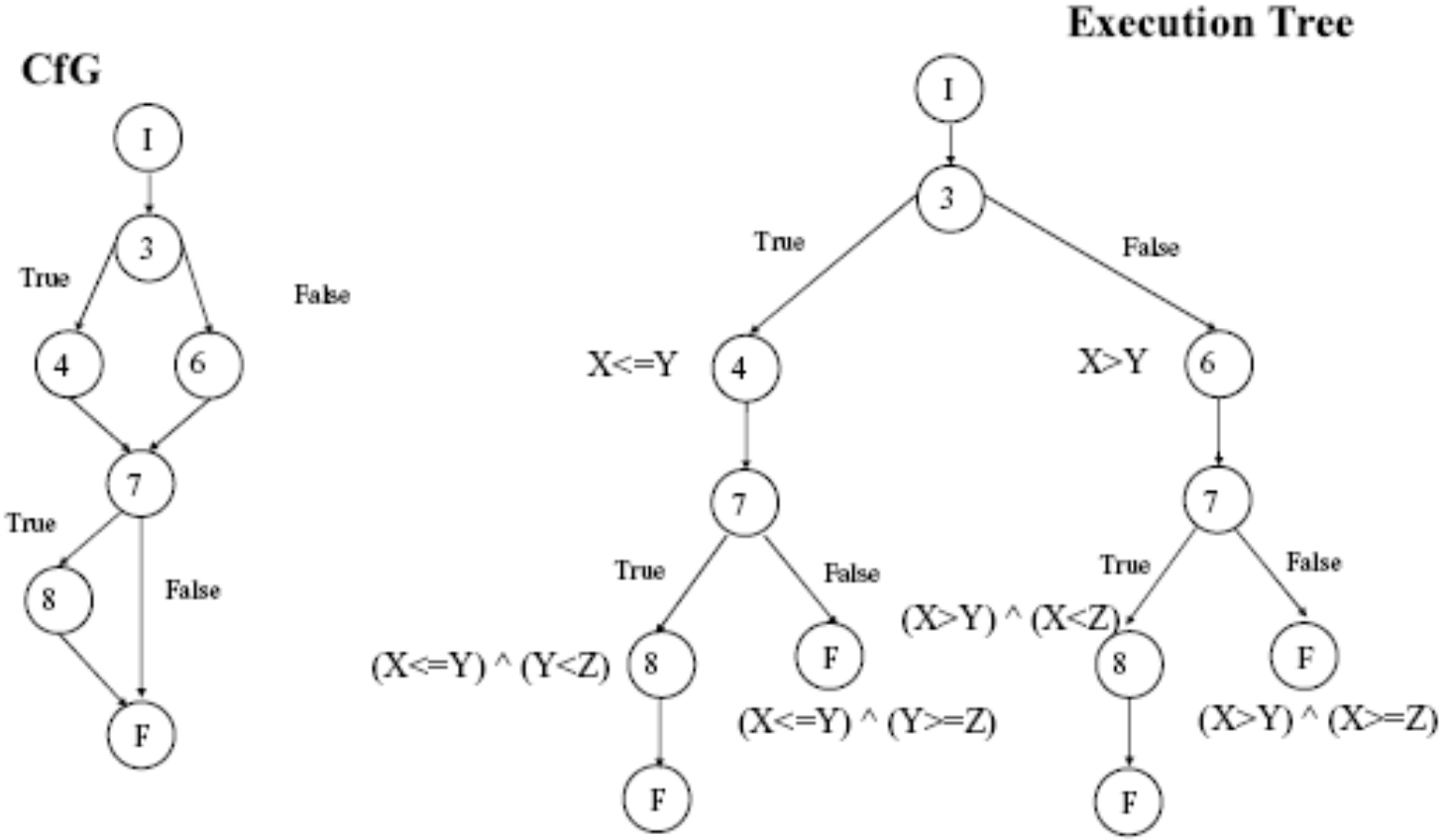
Quali sono le path condition per eseguire gli statement

4

6

8 ?

Path Condition



Ciascuna foglia dello execution tree rappresenta un cammino che sarà percorso per una certa *pc*

Analisi dell'Execution Tree

- Ogni foglia dello execution tree rappresenta un cammino che sarà percorso per certi valori di input
- Le Pc associate a due differenti foglie sono distinte; ciascuna foglia dello execution tree rappresenta un cammino che sarà percorso per la Pc ad essa associata.
- Non esistono esecuzioni per cui sono vere contemporaneamente più Pc (vero solo per programmi sequenziali)

- **Feasible Path**: un cammino per il quale esiste un insieme di dati di ingresso che soddisfa la path condition.
- **Unfeasible Path**: un cammino per il quale non esiste un insieme di dati di ingresso che soddisfa la path condition.
- Se l'output ad ogni foglia è corretto allora il programma è corretto.
- Ma, quanti rami può avere un execution tree?

DETERMINAZIONE DELLA ESEGUIBILITA' DI UN CAMMINO (Path feasibility)

Davis (1973)

Il problema di stabilire se esiste una soluzione per un sistema di diseguaglianze é indecidibile.

- Un cammino é eseguibile se esiste un punto del dominio di ingresso che rende soddisfatta la sua path condition (..un sistema di diseguaglianze).
- La determinazione della feasibility o della infeasibility di un cammino è indecidibile.

NB. se si riesce a dimostrare che ciascun predicato nella path condition è dipendente linearmente dalle variabili di ingresso, allora il problema è risolubile con algoritmi di programmazione lineare.

Appendice

Call Directed Graph

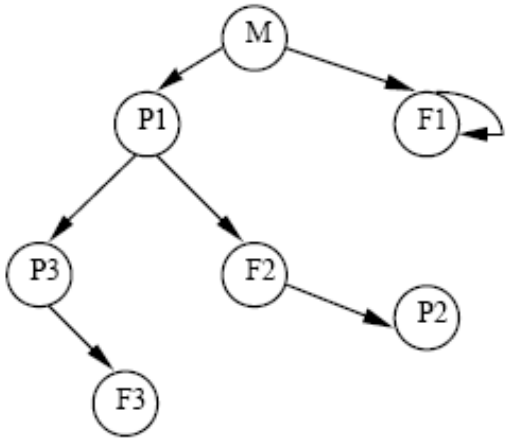
definito da: CDG(P)= (PP, E, s)

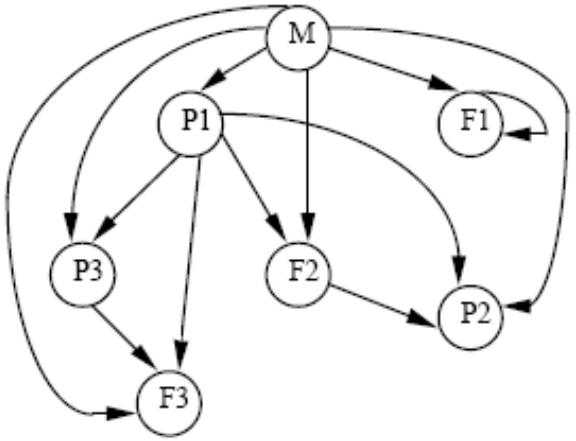
Dove:

PP è l'insieme dei sottoprogrammi di P

E è la relazione di chiamata (s ∪ PP) x PP

s è il main program.





Circuiti e Cappi implicano Ricorsione
Rilevante il concetto di cammino

Raggiungibilità K-fold
Raggiungibilità Totale (Chiusura Transitiva)

Alberi di Dominanza

Dato un GfC(P) o un CDG(P), posto s = nI ovvero s = m, reso aciclico tale grafo (collassando in un unico nodo le sue componenti connesse, quali cappi e circuiti) si ha che:

se *px* e *py* sono due nodi in un grafo aciclico,

- *px* **domina** *py* se e solo se:

  $\forall \mu i(s,py)$, $px \in \mu i$

  dove con $\mu i(s,py)$ si indica un generico cammino da s a py.

- px **domina direttamente** py se e solo se:

(px$\diamond$py) and (px domina py) and ($\forall$ pz: pz$\diamond$px and pz$\diamond$py, se pz domina py allora pz domina px)

Il grafo della riduzione riflessiva e transitiva della relazione di dominanza in CDAG e' un **albero**.

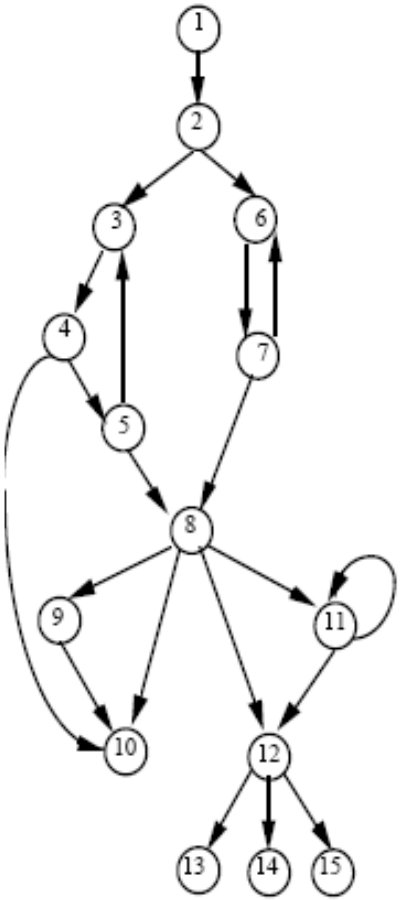
Vi e' un arco (px,py) nell'albero se e solo se px domina direttamente py, e c' é un cammino μ(pu,pv) nell'albero se e solo se pu domina pv.

Alberi di Dominanza

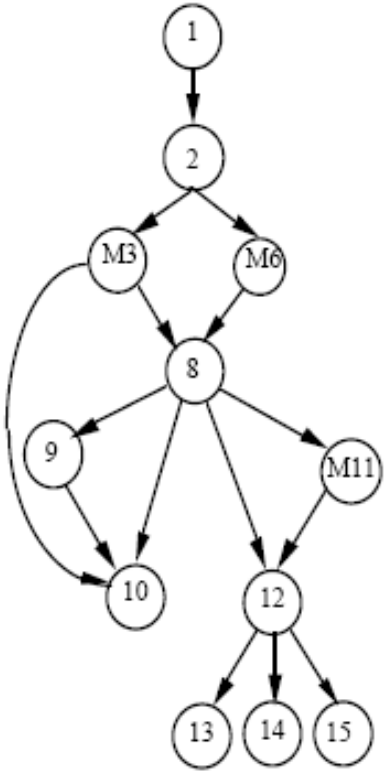
Il nodo s domina tutti gli altri nodi;
la relazione di dominanza è una relazione d'ordine parziale (riflessiva, antisimmetrica, transitiva).

è possibile provare che, eccetto s, ogni nodo ha un unico dominatore diretto.

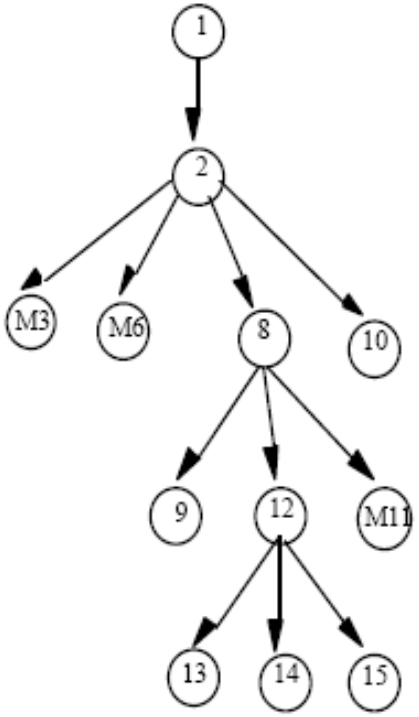
Esempio: albero di dominanza di un dato call-graph



CALL GRAPH



CALL GRAPH ACICLICO

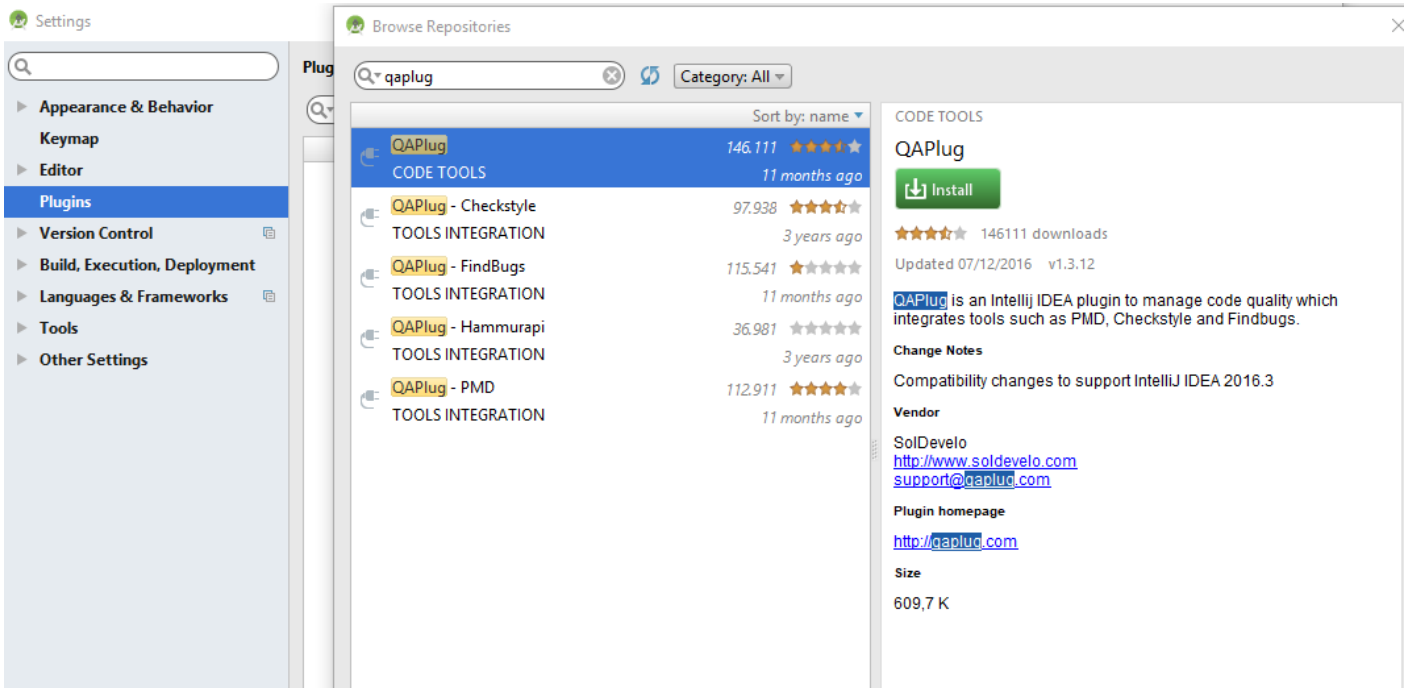


ALBERO DOMINANZE

QAPlug

Esiste un plugin denominato QAPlug che integra le analisi di Lint (native in Android) con quelle Java-oriented di CheckStyle, PMD, Findbugs

Al momento (novembre 2017) non funziona su Android Studio 3.0



Lint (ADT)

Cerca tramite analisi statica nel codice Android *bad smells* appartenenti a 11 c ie diverse:

- Correctness
- Correctness:Messages
- Security
- Performance
- Usability:Typography
- Usability:Icons
- Usability
- Accessibility
- Internationalization
- Bi-directional Text

ad smells sono classificati per gravità:

» Fatal, Error, Warning,Information, Ignore

◢ Correctness:Messages	
ExtraTranslation	Checks for translations that appear to be unused (no default language string)
MissingTranslation	Checks for incomplete translations where not all strings are translated
◢ Security	
HardcodedDebugM	Checks for hardcoded values of android:debuggable in the manifest
PackagedPrivateKey	Looks for packaged private key files
◢ Internationalization	
ByteOrderMark	Looks for byte order mark characters in the middle of files
EnforceUTF8	Checks that all XML resource files are using UTF-8 as the file encoding

Lint: esempio

Rispetto all'app Indovina l'Ora del Delitto, chiamando Lint (Android Tools/Run Lint) vengono segnalati:

0 errors, 10 warnings

Description	Category	Location
Manifest should specify a minimum API level with <uses-sdk android:minSdkVersion="?" />; if it really supports all versions of Android set it to 1.	Correctness	AndroidManifest.xml (IndovinaOraDelitto)
Launcher icons should not fill every pixel of their square region; see the design guide for details	Usability:Icons	hitchicon.jpg in drawable (IndovinaOraDelitto)
[I18N] Hardcoded string "Ipotizza un orario", should use @string resource (3 items)	Internationalization	main.xml:28 in layout (IndovinaOraDelitto)
[I18N] Hardcoded string "Ricomincia da capo", should use @string resource	Internationalization	main.xml:33 in layout (IndovinaOraDelitto)
[I18N] Hardcoded string "Esci", should use @string resource	Internationalization	main.xml:38 in layout (IndovinaOraDelitto)
Missing the following drawables in drawable-xhdpi: icon.png (found in drawable-mdpi, drawable-hdpi)	Usability:Icons	IndovinaOraDelitto
Possible overdraw: Root element paints background @drawable/hitchcock with a theme that also paints a background (inferred theme is @android:style/Theme)	Performance	main.xml:6 in layout (IndovinaOraDelitto)
The resource R.drawable.icon appears to be unused (3 items)	Performance	icon.png in drawable-hdpi (IndovinaOraDelitto)
The resource R.string.hello appears to be unused	Performance	strings.xml:5 in values (IndovinaOraDelitto)
The resource R.string.txtok appears to be unused	Performance	strings.xml:7 in values (IndovinaOraDelitto)

Proposte di progetto

Verificare statisticamente l'esistenza di bad smells in applicazioni Android open source (come quelle presenti su https://f-droid.org/), eseguendo automaticamente lint direttamente da linea di comando (https://developer.android.com/studio/write/lint.html#commandline)

Valutare e provare tecniche per l'implementazione di nuove tecniche di refactoring integrate in Android Studio o in Lint

- http://tools.android.com/tips/lint-custom-rules
- http://tools.android.com/tips/lint/writing-a-lint-check

Risorse per il refactoring

Alcune risorse riguardanti il refactoring:

- Esempi pratici di refactoring
  - *http://ocw.kfupm.edu.sa/user062%5CSWE31601/18_Refactoring.ppt*
- Per divertirsi: un torneo sulla realizzazione di strumenti di refactoring
  - *http://www.metodiagili.it/torneo.html*
- Un'azienda che si occupa di reverse engineering di codice sorgente, refactoring, etc.
  - *http://www.semdesigns.com/Products/DMS/DMSToolkit.html?Home=Refactoring*

CodePro Analytix

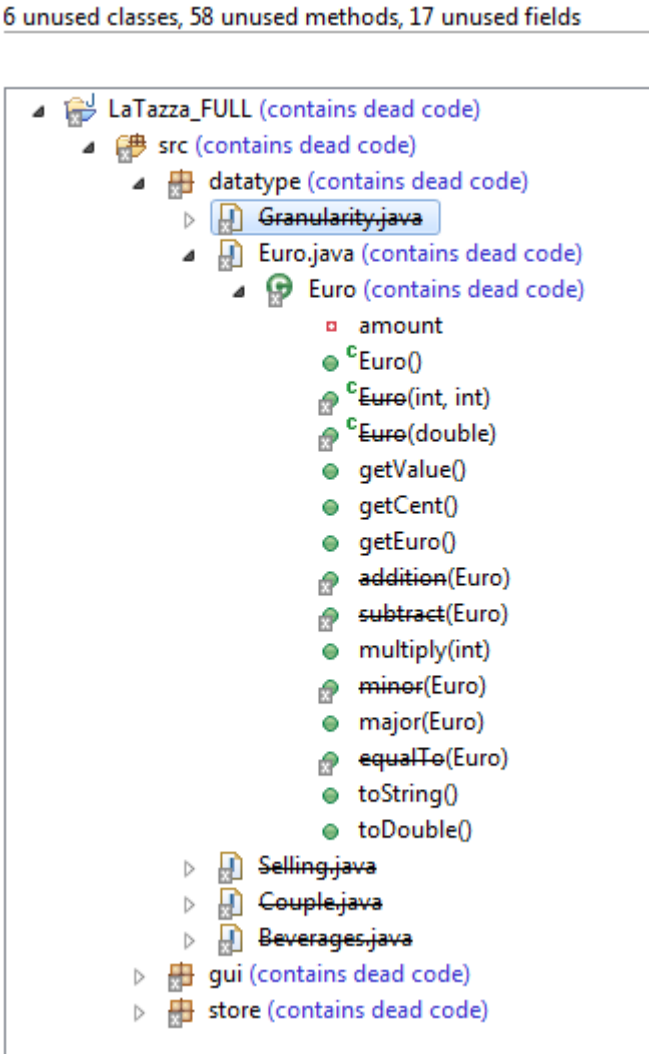
- **Plug-in multifunzionale per Eclipse offerto da Google**
  - https://developers.google.com/java-dev-tools/codepro/doc/
  - Scaricabile direttamente da:
    - http://dl.google.com/eclipse/inst/codepro/latest/3.7
    - Aggiornato ad Eclipse Indigo; dovrebbe funzionare anche per le versioni successive
- **Tutorial e documentazione accessibili da:**
  - https://developers.google.com/java-dev-tools/codepro/doc/

CodePro Analytix Dead Code Analysis

- **CodePro cerca, tramite analisi statica, di verificare quale parte del codice non sia raggiungibile (codice morto)**
  - Le indicazioni di CodePro non possono essere considerate né complete né sempre valide, a causa di possibili utilizzi del codice tramite chiamate dinamicamente generate
- **Per eseguire l'analisi del Dead Code è sufficiente eseguire *Find Dead Code* nel menu contestuale *Code Pro Tools***

Utilizzo di Find Dead Code

- **Selezionare (da proprietà del progetto/*Dead Code Entry Point*) i punti di accesso dell'applicazione (di solito il main)**
- **Avviare *Find Dead Code***
- **Osservare i risultati nella view *Dead Code***
  - Le classi/metodi/attributi barrati sono inutilizzabili (secondo quest'analisi)



Utilizzo di Find Dead Code

- **La ricerca del Dead Code non funziona bene in caso di codice generato dinamicamente**
  - Un caso tipico è rappresentato dal codice ascoltatore di eventi utente
    - *In questo caso, è opportuno considerare le classi che rispondono agli ascoltatori come ulteriori Entry Point*
    - *Per conoscere il Dead Code interno ai metodi, può essere opportuno segnare ogni metodo come possibile entry point*