

# ANALISI STATICA

PROGRAMMAZIONE AD OGGETTI

C.D.L. INGEGNERIA E SCIENZE INFORMATICHE

**Danilo Pianini** — [danilo.pianini@unibo.it](mailto:danilo.pianini@unibo.it)

**Roberto Casadei** — [roby.casadei@unibo.it](mailto:roby.casadei@unibo.it)

 [versione stampabile](#)

# Analisi di qualità del codice sorgente

## “Software di analisi statica del codice” - Definizione

Software in grado di analizzare il codice sorgente per individuare:

- Potenziali bug, magari dovuti a distrazione
- Possibili miglioramenti, ottimizzazioni o pratiche difformi da quelle consigliate
- Codice duplicato, segnale di una progettazione discutibile
- Stile non conforme

## USO

L'analisi automatica del proprio codice consente di migliorare la qualità del codice, aiuta ad apprendere il modo corretto di scrivere, riduce il costo di manutenzione, e garantisce uniformità fra le parti sviluppate da persone diverse (aiutando quindi il sistema di controllo versione!).

## Code checking

I software che vedremo sono eseguibili in due modalità:

- *Stand-alone*: il software viene eseguito e genera un report
- Come *plug-in*: il software viene integrato con l'IDE (ad es. VS Code), e segnala i problemi sotto forma di warning
- Come *plugin Gradle*: Gradle viene configurato per eseguire il software nella prima modalità

Noi vedremo la terza modalità.

# SpotBugs (ex FindBugs)}

SpotBugs scansiona il *bytecode* generato dal compilatore, e dalla sua analisi cerca di scoprire potenziali bug nel sorgente, ad esempio:

- Uguaglianza esatta fra *float* o *double*
- Utilizzo di *==* invece di *equals()*
- Mancata annotazione di annotazioni usate a runtime via reflection
- Uso errato di meccanismi di sincronizzazione
- Assenza di copie difensive
- Variabili non utilizzate
- Vulnerabilità di sicurezza
- Tanti altri! Si veda: <https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html>

# PMD e CPD

PMD si occupa di trovare imperfezioni nel codice:

- Campi pubblici, protetti o default
- Mancato uso di final
- Singular fields
- Integra CPD (copy/paste detector) per verificare se vi siano blocchi di codice copincollati
- Tanti altri! Si veda: [https://pmd.github.io/latest/pmd\\_rules\\_java.html](https://pmd.github.io/latest/pmd_rules_java.html)

# Checkstyle

Cos'è: Checkstyle si occupa di trovare errori di stile:

- Mancanza di commento Javadoc
- Spaziature non corrette
- Parentesi assenti
- Magic numbers
- Altro: <http://checkstyle.sourceforge.net/checks.html>

# Configurazione e utilizzo tramite Gradle

Ciascun tool di quelli introdotti ha un proprio plugin gradle che consente di attivarlo e configurarlo. Per gli scopi del corso, è stato sviluppato un plugin che applica i tre plugin precedenti, preconfigurandoli in modo “ragionevole”.

```
plugins {  
    id("org.daniopianini.gradle-java-qa") version "0.40.0"  
}
```

di default, questo causa un fallimento della build se ci sono errori in analisi statica:

- la configurazione è *aggressiva*.
- Plugin
- Aggressività
- Build reports

# Falsi positivi e falsi negativi

<i>Osservato</i> \ <b>Reale</b>	<b>Vero</b>	<b>Falso</b>
<i>Vero</i>	Positivo	Falso positivo
<i>Falso</i>	Falso negativo	Negativo

- **Sensibilità:** capacità di un test di identificare i casi veri
  - ▶ Alta sensibilità  $\Rightarrow$  pochi falsi negativi
  - ▶ Un test molto sensibile tende a sbagliare “in eccesso”, preferendo i falsi positivi ai falsi negativi.
- **Specificità:** capacità di un test di identificare i casi negativi
  - ▶ Alta specificità  $\Rightarrow$  pochi falsi positivi
  - ▶ Un test molto sensibile tende a sbagliare “in difetto”, preferendo i falsi negativi ai falsi positivi.

Sensibilità e specificità spesso vanno *bilanciate*, raramente in casi reali un test può essere sia molto sensibile che molto specifico

**CI SARANNO CASI IN CUI L'ANALISI STATICA DARÀ DEI FALSI POSITIVI**

(ci saranno anche falsi negativi, ma è più difficile trovarli...)



# Gestire falsi positivi: soppressione

La *ragione* per cui un caso di problema sia da classificare come *falso positivo* va *studiata*

- Potrebbe essere, in effetti, un *bug* o una limitazione del tool di analisi statica
- Potrebbe essere che il difetto ci sia veramente, *ma non dipenda da noi* (magari stiamo usando una libreria esterna che ci forza a fare le cose così)
- Potrebbe essere che il difetto sia presente, *ma sia voluto*
  - ▶ Abbiamo scientemente fatto una scelta di design, intendiamo mantenerla e sappiamo giustificarla
- Potrebbe essere che il difetto sia presente, *ma l'alternativa sia peggiore*
  - ▶ caso specifico non supportato dal software
  - ▶ impossibilità di utilizzo di una alternativa
  - ▶ performance troppo basse (raro nei progetti di OOP...)

In questi casi, si *disabilita puntualmente l'analisi statica nel punto in cui è presente il falso positivo*

Questa operazione è detta **soppressione** e va *sempre giustificata*

# Soppressione in Checkstyle

- La soppressione in checkstyle avviene tramite commenti, ed è configurabile
- Nella configurazione fornitavi, si sopprime con delle “comment fences”:

```
// CHECKSTYLE: <ruleName> OFF  
<java code with the false positive>  
// CHECKSTYLE: <ruleName> ON
```

- dove **<ruleName>** va sostituito col nome della regola di Checkstyle che si sta violando

È bene aggiungere un commento che spieghi la ragione della soppressione

```
// CHECKSTYLE: <ruleName> OFF  
// Rule disabled due to false positives, see this bug report: https://...  
<java code with the false positive>  
// CHECKSTYLE: <ruleName> ON
```

# Soppressione in PMD

- La soppressione in PMD avviene tramite commento in linea

```
<java code with the false positive> // NOPMD suppressed as it is a false positive
```

Se la linea di codice diventa troppo lunga, si può spezzare:

```
<java code with the false positive> // NOPMD  
// suppressed as it is a false positive
```

# Soppressione in Spotbugs

Spotbugs lavora sul *bytecode*, quindi non può essere soppresso usando dei commenti  
Occorre invece usare una speciale **annotazione**, contenuta nella libreria:

- **com.github.spotbugs:spotbugs-annotations**

E quindi, in **build.gradle.kts**

```
dependencies {  
    compileOnly("com.github.spotbugs:spotbugs-annotations:4.7.3") // Use the latest version  
}
```

Sarà a questo punto disponibile l'annotazione **@SuppressFBWarnings**:

```
@SuppressFBWarnings(  
    value = { // List of bugs to be suppressed  
        "BC_UNCONFIRMED_CAST_OF_RETURN_VALUE",  
        "RV_RETURN_VALUE_IGNORED_BAD_PRACTICE"  
    }, // String with the reasons for them to be suppressed  
    justification = "A ChoiceDialog is always in its own stage"  
        + ", and we don't need the status of the Runnable"  
)  
<java code with the false positive>
```

