

# BUILD SYSTEMS E INTRODUZIONE A GRADLE

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

# La costruzione del software

Costruire sistemi software non è solo *programmare*. Dipendentemente dal sistema in esame, potrebbero servire:

- Manipolazione e pre-processing del sorgente (inclusa generazione)
- Verifica della qualità del sorgente
- Gestione delle dipendenze
  - ▶ Ricerca, scaricamento, e importazione delle librerie
- Compilazione
- Manipolazione del binario compilato
- Esecuzione dei test
- Misurazione della qualità dei test (e.g., coverage)
- Generazione della documentazione

In principio, si può anche fare a mano

- ma richiederebbe **molto tempo**...
- ...e gli umani si stancano presto di lavori noiosi e ripetitivi

# Build automation

Automatizzazione del processo di costruzione del software

- Di fatto, scrivere software che di lavoro fa manutenzione di altro software

## Stili

- **Imperativo/Personalizzato**
  - ▶ Tipicamente realizzato tramite script in qualche linguaggio di programmazione
  - ▶ *Flessibile* e *configurabile*
  - ▶ Difficile da adattare e *riusare*
- **Dichiarativo/Standardizzato**
  - ▶ Tipicamente realizzato tramite un file di configurazione di un software dedicato alla build automation
  - ▶ *Portabile* e di semplice comprensione
  - ▶ *Limitato* dalle opzioni di configurazioni disponibili, e quindi poco flessibile

# Convention over configuration

Principio per cui un certo sistema software ha una configurazione “ragionevole” di default, che può essere sovrascritta in caso di necessità

- Induce la creazione di *standard di fatto*
  - ▶ La convenzione tende a diventare il modo “normale” di fare le cose per minimizzare la configurazione
- Riduce le *ripetizioni*
- Aumenta la portabilità!

# Automatori ibridi

Sono sistemi che cercano di unire il meglio dei sistemi dichiarativi e imperativi

- Il file di configurazione è in realtà uno *script* in un linguaggio di programmazione vero e proprio
- Aprendolo sembra un file di testo con la configurazione
- In realtà è uno script valido!
- Quanto non specificato si assume come da convenzione
- Quando si vuole personalizzare qualcosa, si ha a disposizione la “potenza di fuoco” di un linguaggio di programmazione vero e proprio

## Esempi

- Sbt, che si appoggia su Scala
- Gradle, che si appoggia su Kotlin o Groovy

Il linguaggio host deve consentire di costruire dei **Domain-Specific languages**

- ossia, essere così flessibile da permettere di costruire un “linguaggio nel linguaggio”

# Gradle

- Un **moderno** build system ibrido
  - ▶ Pilotato in Kotlin (preferibile) o in Groovy
- Supporta Java (oltre a C/C++, Scala, Kotlin...)
  - ▶ In tutto l'ecosistema, che include Android
- Ne vedremo solo le basi di utilizzo
  - ▶ Per noi è strumentale a costruire software Java
  - ▶ Impareremo come sfruttarlo per automatizzare le operazioni di cui sopra

**Gradle è in espansione: Google trends**

# Concetti base in Gradle: **task**, **progetto**, **plugin**

## Progetto

Una directory contenente il file speciale **build.gradle.kts** e/o **settings.gradle.kts**, detti *build file*. La loro presenza segnala a Gradle che la cartella rappresenta un **progetto**

## Plugin

Componente software contenente **task** pronti all'uso. Gradle contiene diversi plugin pronti all'uso (per i linguaggi più comuni, come Java).

## Task

Un **task** in Gradle rappresenta una *singola operazione atomica* del processo di costruzione del software

- *singola* → un task fa una sola cosa (Single Responsibility Principle)
- *atomica* → indivisibile: un task comincia e finisce senza interruzione

Qualunque esecuzione di Gradle richiede di specificare uno o più **task**, ad esempio:

- **gradle tasks** (elenca i task disponibili, escludendo quelli non categorizzati)
- **gradle tasks --all** (elenca tutti i task disponibili)
- **gradle compileJava** (compila i sorgenti java)

Gradle è in grado capire le *dipendenze* fra task ed eseguirli nell'ordine corretto.



# Gradle: configurazione minimale per Java

- Gradle viene pilotato con due file:
  - ▶ **settings.gradle.kts**
    - ▶ Per i nostri scopi, serve solo a dare un nome al progetto
  - ▶ **build.gradle.kts**
    - ▶ Conterrà tutta la logica di costruzione del software
    - ▶ Ma noi sfrutteremo le convenzioni, configurando ben poco!
- Al momento, ci basta una sola riga di codice per ciascuno!

## settings.gradle.kts

```
rootProject.name = "minimal-build"
```

## build.gradle.kts

```
plugins { java }
```

Così configurato, Gradle autonomamente:

- cerca e compila i sorgenti java dalla cartella: **src/main/java**
- produce i binari dentro: **build/classes/java/main**

Vogliamo percorsi diversi? Va configurato.

# Gradle: Hello World in Java, struttura

```
├── build.gradle.kts
├── settings.gradle.kts
└── src
    └── main
        └── java
            └── HelloWorld.java
```

**build.gradle.kts**

```
plugins { java }
```

**settings.gradle.kts** (opzionale)

```
rootProject.name = "hello-world"
```

**HelloWorld.java**

```
public class HelloWorld {
    public static void main(String... args) {
        System.out.println("Hello, world!");
    }
}
```

# Gradle: Hello World in Java, task e loro utilizzo

- elencare i task disponibili:
  - ▶ `gradle tasks --all`
- compilazione:
  - ▶ `gradle compileJava`
- pulizia (cancellazione della directory `build` dove Gradle lavora):
  - ▶ `gradle clean`
- esecuzione (non responsabilità di Gradle):
  - ▶ `java -cp build/classes/java/main HelloWorld`

# Il build system come dipendenza

**Problema:** come tutti i software *anche il build system cambia*

Se da una versione all'altra di Gradle dovesse cambiare la convenzione, cosa succederebbe?

- Il nostro software smette di funzionare se aggiorniamo il build system!

E se avessimo *progetti diversi* che richiedono *versioni diverse* di Gradle?

## CI SERVE UN MODO PER:

1. avere sempre *la giusta versione* di Gradle
  - ▶ Potenzialmente ogni progetto ha la sua
2. fare in modo che più versioni di Gradle coesistano senza “darsi fastidio”
  - ▶ Non vogliamo certo scaricare e installare versioni diverse a seconda del progetto su cui stiamo lavorando
3. già che ci siamo, sarebbe carino se questo sistema sapesse scaricare e installare Gradle, senza richiedere multiple installazioni manuali all'utente

# Gradle wrapper

Un insieme di script con un software minimale che:



1. Scarica la versione di Gradle indicata in un file di configurazione
  - ▶ se non già disponibile nel sistema
2. Usa quella versione per costruire il nostro sistema software!

Il wrapper può (deve) esser copiato in ogni progetto che gestiamo con Gradle


Dato che il wrapper sa come scaricare ed installare Gradle, non occorre scaricare ed installare gradle manualmente

- Anche se è comodo, la versione di Gradle installata può generare le versioni wrapper

# Progetti Gradle con wrapper




1. Script bash eseguibile (/🍏): **gradlew**
2. Script batch eseguibile (): **gradlew.bat**
3. File di configurazione con indicata la versione di Gradle:  
**gradle/wrapper/gradle-wrapper.properties**
4. Software Java che scarica la versione di Gradle descritta nel file di configurazione:  
**gradle/wrapper/gradle-wrapper.jar**

Wrapper pronto per esser scaricato:

- <https://github.com/DanySK/Gradle-Wrapper/archive/refs/heads/master.zip>
- **Attenzione:** su  e 🍏, eseguire anche il comando **chmod +x gradlew** per rendere eseguibile lo script
- setta i permessi Unix per eseguire, il cui valore viene resettato dalla compressione in formato zip
- alternativamente, lo script va eseguito chiedendo all'interprete della linea di comando di interpretarlo
- **sh gradlew**

# Utilizzo di Gradle con wrapper

Se abbiamo il Gradle wrapper configurato in un progetto, possiamo usarlo attraverso uno dei due script:

- **gradlew** (/) o **gradlew.bat** () , a seconda della nostra piattaforma
- seguito dall'elenco dei **task**

## NOTA

Su sistemi  e  per eseguire lo script occorre anche includere il percorso corrente:

- **./gradlew**

o, in alternativa, chiedere all'interprete dei comandi di eseguire

- **sh gradlew**
- **bash gradlew**

