

Un Framework per la Meta-programmazione in *Minecraft*

Libreria OOPACK

Nanni Alessandro

Alma Mater Studiorum Università di Bologna

16/12/2025

Contenuti

- Minecraft come piattaforma di sviluppo 2
- Limiti di *mcf*function 3
- Libreria OOPACK 5
- Architettura del software 6
- Metodi di utilità 8
- Working Example 9
- Risultati e metriche 11

Minecraft come piattaforma di sviluppo

- Minecraft è un ambiente programmabile tramite:
 - *Datapack* per la logica
 - *Resourcepack* per le risorse
- Linguaggio: *mcfuction* (DSL interpretato) + file JSON
- **Obiettivo della tesi:**
 - Analizzare le problematiche del DSL
 - Progettare una libreria Java per la meta-programmazione
 - Semplificare lo sviluppo di *pack*

Limiti di *mcf*function

- **Mancanza di strutture avanzate:**
 - Nessuna variabile o struttura dati complessa
 - Solo operazioni su interi
- **Frammentazione del codice:**
 - Ogni funzione richiede un file separato
 - Assenza di *code blocks*
- **Elevato boilerplate:**
 - Fino a 7 file per definire un oggetto semplice
- **Gestione matematica limitata:**
 - Necessità di *lookup table* per funzioni come seno, coseno, radice quadrata.

Esempio di *Lookup table* per \sqrt{x} , con $0 \leq x \leq 100$.

```
1  data modify storage my_storage sqrt set value [  
2      0,  
3      1.0,  
4      1.4142135623730951,  
5      1.7320508075688772,  
6      2.0,  
...                                     ...  
102    10.0  
103    ]
```

Libreria OOPACK

Astrazione del *pack* come albero di oggetti tipizzati.

- **Workflow:**

1. Scrittura codice ibrido Java + mcfuction
2. Validazione statica della struttura
3. Generazione automatica dei file tramite `build()`

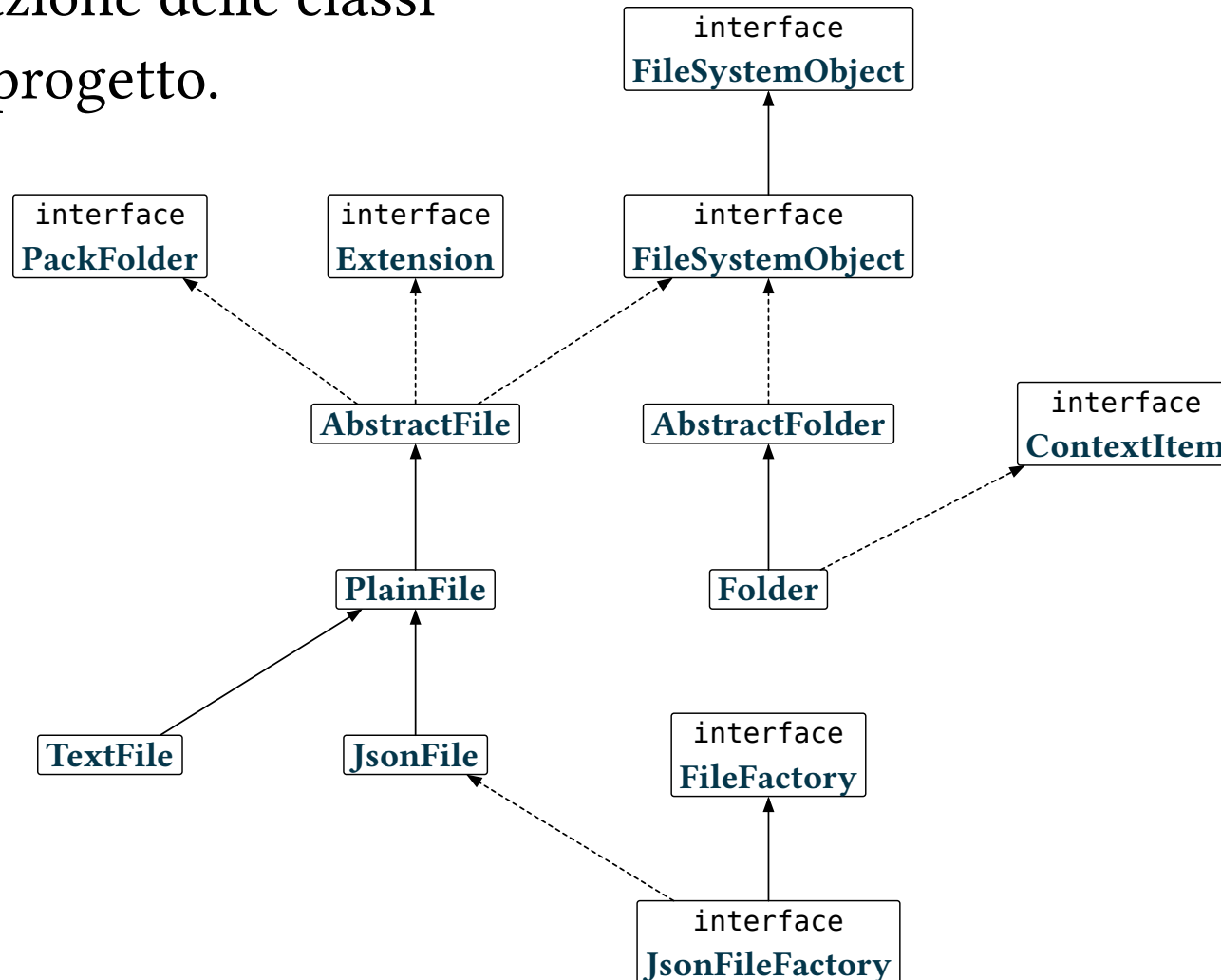
- **Vantaggi:**

- Utilizzo di costrutti di alto livello
- Riduzione del codice ripetitivo
- Semplificazione della struttura del progetto

Architettura del software

- **Design Pattern utilizzati:**
 - **Composite:** Gestione gerarchica di file e cartelle
 - **Factory:** Istanziamento controllata degli oggetti
 - **Builder:** Configurazione flessibile del progetto
- **Sistema basato su interfacce:**
 - `Buildable` → oggetto costruibile
 - `FileSystemObject` → l'oggetto ha dei contenuti
 - `PackFolder` → per indicare se l'oggetto è di tipo *data* o *resource*
 - `Extension` → per indicare l'estensione del file

Rappresentazione delle classi astratte del progetto.

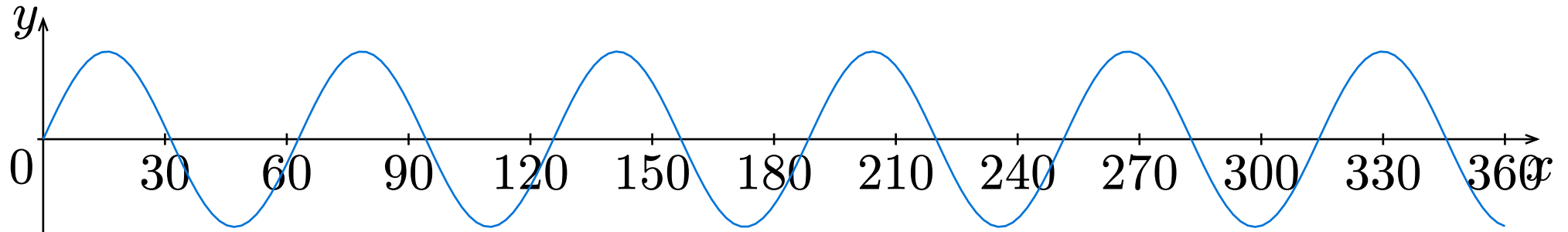


Metodi di utilità

- **Metodo `find()`:**
 - Ricerca file specifici nel progetto
 - Istanziamento automatico se non esistente (*lazy loading*)
- **Funzionalità di alto livello:**
 - `addTranslation(key, value)` per localizzazione
 - `addSound()` per registrazione audio
 - `setOnTick()` per le funzioni da eseguire ogni *game loop*.
 - `setOnLoad()` per le funzioni da eseguire ad ogni ricarica del gioco.

Working Example

- **Obiettivo:** modificare un oggetto renderlo in grado di produrre un'onda sinusoidale la cui distanza dipende dalla munizione.



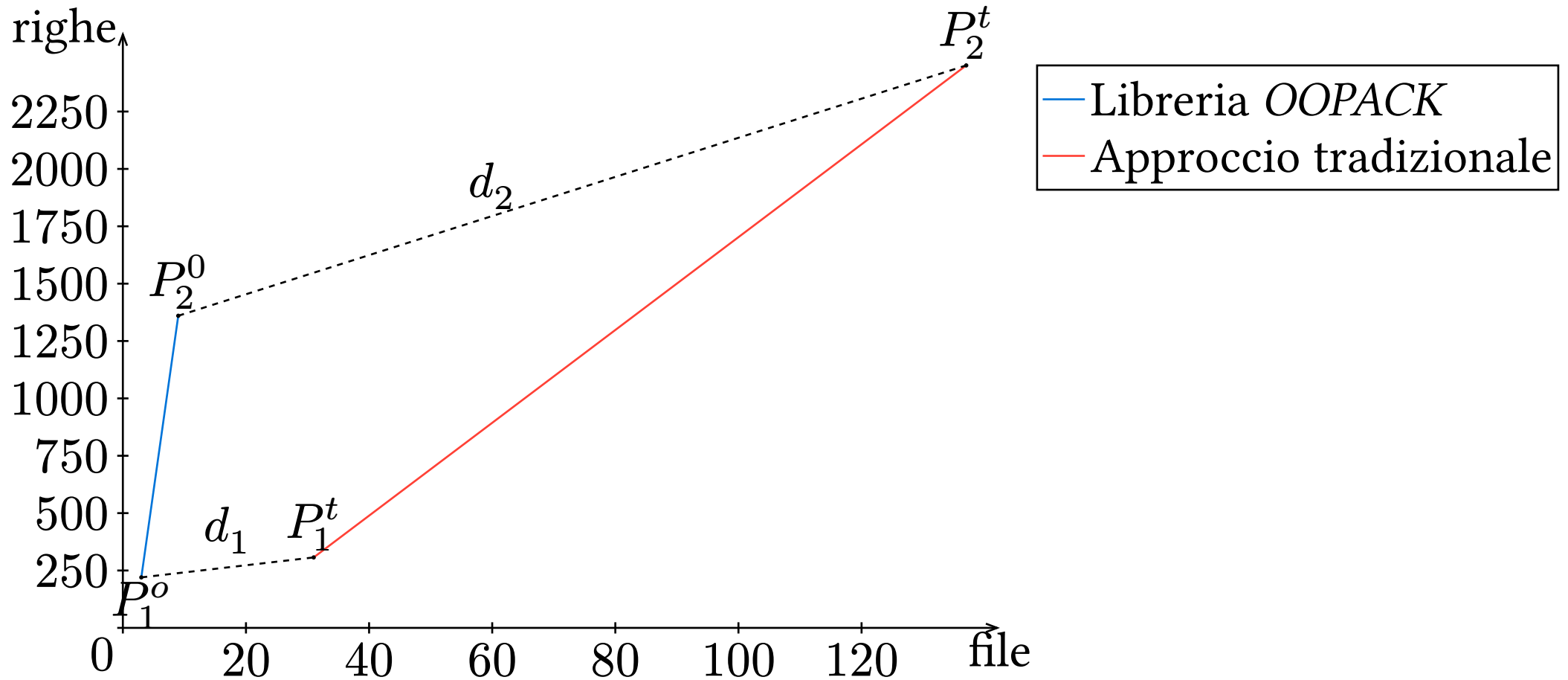
Metodo scritto per facilitare la creazione di boilerplate per le munizioni.

```
make("blue_ammo", "Munizione Blu", "Blue Ammo",  
    "diamond", 20);
```

Generazione e inizializzazione della *lookup table* richiesta:

```
private void makeSinLookup() {  
    StringBuilder sin = new StringBuilder("data modify  
storage esempio:storage sin set value [");  
    for (int i = 0; i <= 360; i++) {  
sin.append("{value:").append(Math.sin(Math.toRadians(i *  
10))).append("},");  
    }  
    sin.append("]");  
    Util.setOnLoad(Function.f.of(sin.toString()));  
}
```

Risultati e metriche



- **Efficienza della generazione:**
 - Progetto P_1 : 1 file sorgente \rightarrow 10,3 file output
 - Progetto P_2 : 1 file sorgente \rightarrow 15,2 file output
- **Automazione crescente:**
 - Maggiore scala \rightarrow maggiore automazione
 - Distanza $d_1 = 91,4$ vs $d_2 = 1098,5$
 - A densità di codice raddoppiata, beneficio d aumenta di 12 volte

Grazie per l'attenzione