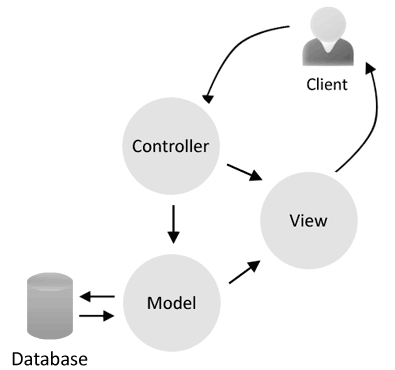


Base Di Progetto:

Il nostro progetto del gioco “MasterMind”, sviluppato in Java e in seguito Esteso a Gradle 5.5, è stato realizzato seguendo uno dei Pattern di Sviluppo più comuni e più efficienti nell’ambito della Programmazione, quello MVC (Model, View, Controller).

Inoltre, onde evitare che il software diventasse una matassa ingarbugliata, detta anche “Spaghetti con Polpette”, durante la scrittura del codice, abbiamo passo passo, cercato di rispettare il più possibile i principi SOLID della programmazione quali:

* Singola Responsabilità delle Classi.
* Codice chiuso alle modifiche, ma aperto alle Estensioni.
* Barbara Liskov Substitution.
* Segregazione delle Interfacce.
* Dipendenze Inverse.

Pattern MVC:

Model-View-Controller (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte, quali:

1. Model.
2. View.
3. Controller.

Nel nostro caso, abbiamo realizzato appositamente 3 classi distinte che avrebbero poi rappresentato i 3 blocchi del Pattern:

1. ConsoleGame
2. InputOutput
3. MatchCoordinato

ConsoleGame:

Classe che fornisce i metodi per accedere ai dati utili all'applicazione, che, non avendo un database, utilizza classi apposite, come GameParameters e Campo, dove vengono salvati i parametri di ciascuna partita.

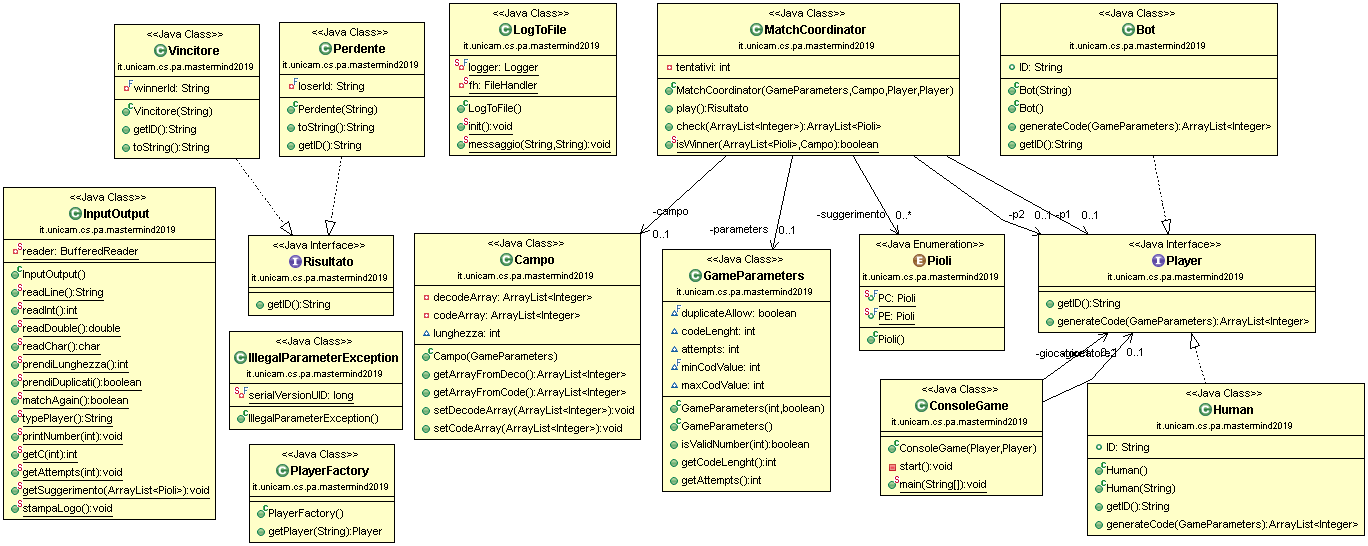
InputOutput:

Classe con lo scopo di prendere i dati in input dall’utente e stampare gli output in console, così da far interagire l’utente con il programma.

MathCoordinator:

Classe che per comodità chiameremo Arbitro, perché essenzialmente il suo compito è quello di fare da tramite tra il Model e la View, da quest’ultima, prenderà i comandi, passati dall’utente, e andrà di conseguenza a modificare lo stato delle altre due componenti.

UML:

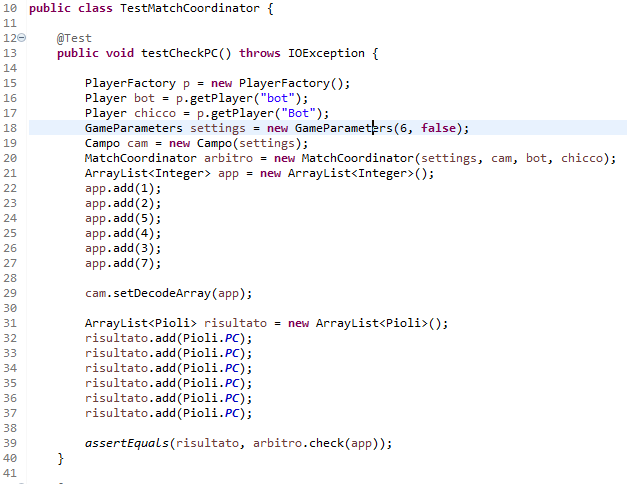


Estendibilità:

Rispettando i principi SOLID e il Pattern MCV per la progettazione del software, rimarrà semplice in un futuro estendere il progetto, ad esempio con un’interfaccia grafica, dove basterà sostituire la classe InputOutput con delle classi apposite come quelle fornite da JavaFX o una classe dove viene implementata una Socket, così da rendere l’applicazione un server remoto, dove i giocatori si connetteranno e si interfacceranno con il programma.

Un altro esempio di Estendibilità del codice è quello di implementare un giocatore con una precisa strategia, come quella di Donald Knuth, che risolve il MasterMind in circa 5 mosse. L’aggiunta quindi, di un’intelligenza artificiale, non sarà un ostacolo in quanto, attraverso la classe PlayerFactory, basterà aggiungere un nuovo tipo di Giocatore e la relativa Classe AI.

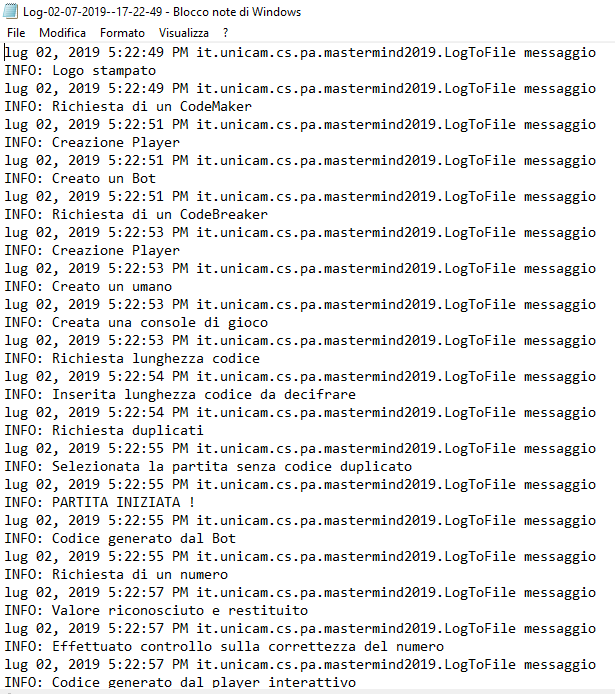
Test Junit 4:

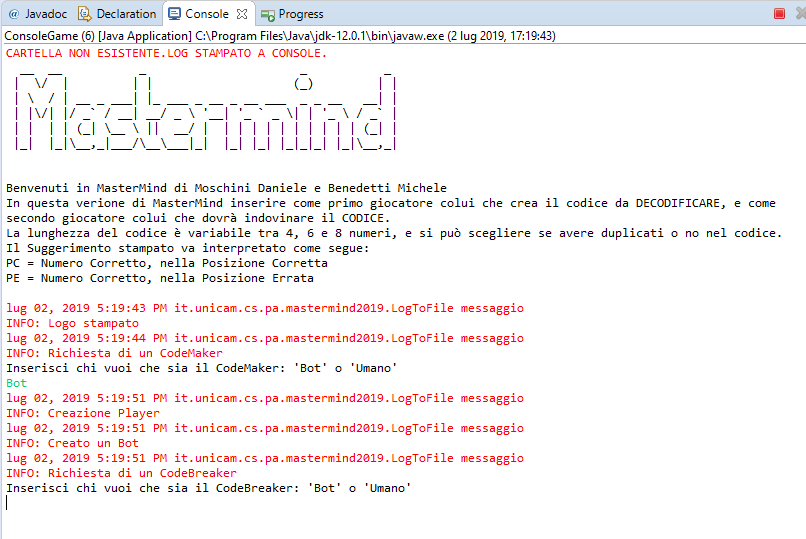
Abbiamo inoltre implementato una serie di test in Junit4 per verificare automaticamente la correttezza del programma, al variare del codice in futuro. Sa scelta dei test eseuiti, è basata sull’importanza delle Classi e dei relativi Metodi, ed il loro impatto nel programma.

Una delle classi principale è appunto il MatchCoordinator, il quale, ha il compito di Gestire la Partita, con la conseguente individuazione del Vincitore o Perdente.

Qui affianco riportiamo una porzione di codice dei Test eseguiti per verificare il corretto funzionamento della Classe.

Logging:

Per esigenza abbiamo implementato la funzionalità di Logging, messa a disposizione dalla libreria Java.util.logging che ci ha permesso di tenere traccia di ciò che il programma fa durante l’esecuzione, e al termine di esso, va a scrivere tutte le informazioni acquisite in un File di tipo .log nella directory C:/Log, che se inesistente, comporterà una stampa a console dei messaggi di Log.



Gradle:

Gradle è un software open-source che permette l’automazione dello sviluppo dei progetti, come ad esempio nel nostro caso, in Java.

Lo abbiamo aggiunto in seguito a costo zero, attraverso il comando grdle init lo abbiamo inizializzato, selezionando quindi i vari linguaggi e formattazioni utilizzate nel nostro progetto, tra cui:

1. Tipo di Progetto: Java Application
2. Build Script DSL: Groovy
3. Test Framwork: Junit

E abbiamo lasciato a lui il compito di generare per noi i seguenti file:

1. [Javadoc](../MM/build/docs/javadoc/index.html)
2. [Risultati\_Test](../MM/build/reports/tests/test/index.html)

Conclusioni:

Alla fine dello sviluppo, a causa delle tempistiche, abbiamo rinunciato all’implementazione della Classe AI, e ad una possibile interfaccia grafica in JavaFX.

Per quanto riguarda il resto del progetto sino ad ora, rispetta il corretto funzionamento del gioco MasterMind, e delle relative responsabilità delle classi nel progetto, con l’aggiunta a costo zero di Gradle 5.5.