

Maintenance

Versione	Data	Descrizione
1.0	19 Gennaio 2026	Attività svolte con plugin PMD e STAN

Attività di refactoring

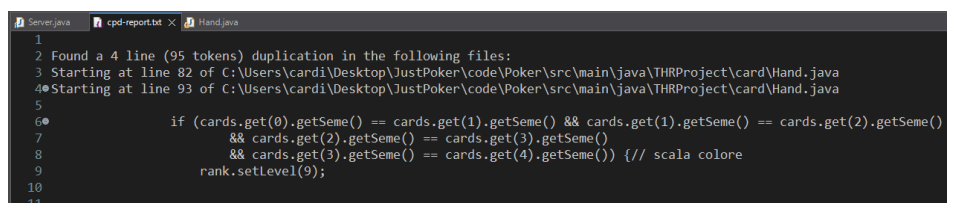
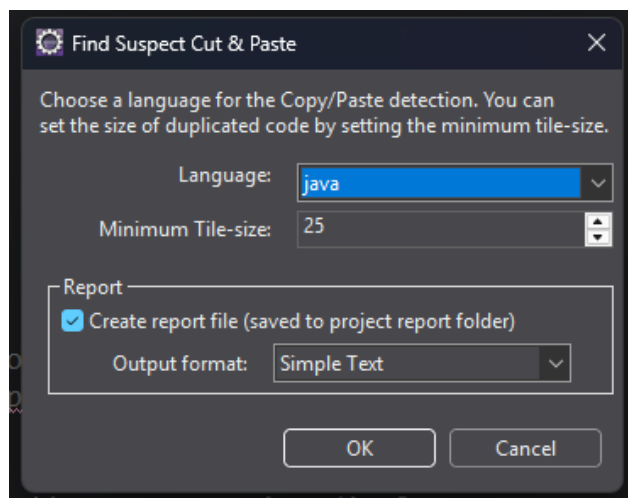
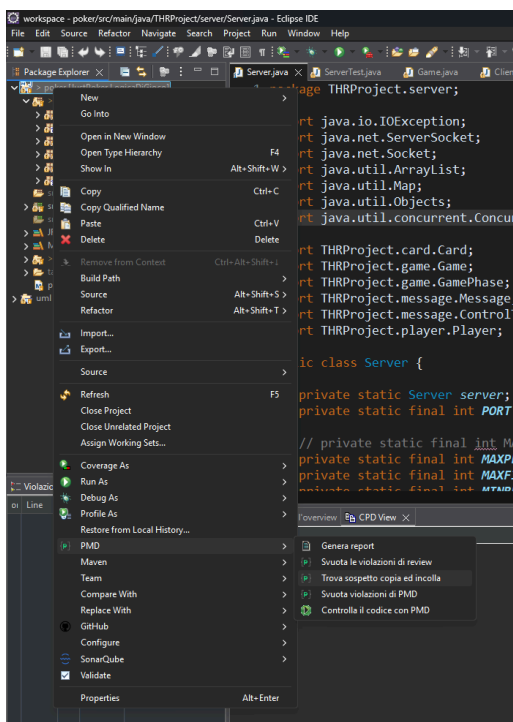
Dopo il completamento dello sviluppo del codice ci concentreremo su diverse attività di refactoring, ovvero modifiche che non portino cambiamenti a livello funzionale, ma solo a livello di design. Per farlo utilizzeremo diversi tool a scopi diversi.

PMD

Il tool PMD, integrato nell'IDE Eclipse, ci permette di effettuare una scansione completa del codice per trovare righe duplicate. Il plugin genera un documento di report visibile nella directory del progetto, ovvero una linea guida che indica esattamente dove sono i problemi nel codice, specificando Classi e Righe. Dopo aver analizzato al meglio quali fossero i problemi di nostro interesse (non tutti i duplicati erano problematici) abbiamo modificato il codice:

- accorpato in un'interfaccia Communicator il metodo sendMessage(Object obj) che veniva utilizzato da Client e da ClientHandler (Server)
- creato il metodo isActive() per PlayerStatus che facilita il controllo dello stato "attivo" (no fold e no end) del Player
- creato il metodo valorePuntataValido() per controllare che la puntata del Player fosse consona ai valori del suo stato
- creato metodo getPossibleTurns() che restituisce gli id di turno dei soli giocatori attivi
- creato metodo isColore() per controllare che la mano abbia come punteggio "colore"

Sotto mostriamo come abbiamo agito:



STAN

Grazie all'utilizzo del plugin STAN calcoliamo le metriche della qualità del codice del nostro progetto.

- Package card:

Artifact	Metric	Value	
.card	D	-1	🚩
.card.Hand.checkRank()	CC	26	❌
.card.Hand.checkRank()	ELOC	106	🚩

- la misura della complessità ciclomatica [CC] e della lunghezza [ELOC] del metodo checkRank() della classe Hand non rispettavano i valori di soglia: è stato quindi eseguito un refactoring del metodo, separandolo in diversi metodi e rendendolo più chiaro e manutenibile
 - la misura della distanza [D] indica che il package è completamente concreto e stabile, infatti non ha classi astratte e ha solo classi afferenti: ciò è giustificato dal fatto che il suo contenuto modella concretamente quelli che sono gli oggetti specifici del dominio. Per migliorare la situazione è stato diviso in due package, uno di logica e uno di modello.
- Package client:

Artifact	Metric	Value	
.client.ServerListener.run()	CC	23	❌
.client.ServerListener.run()	ELOC	68	🚩

- la misura della complessità ciclomatica [CC] e della lunghezza [ELOC] del metodo run() della classe ServerListener non rispettavano i valori di soglia: è stato quindi eseguito un refactoring del metodo, separandolo in diversi metodi grazie ad un dispatcher e rendendolo più chiaro e manutenibile
- Package server:

Artifact	Metric	Value	
.server.Server	ELOC	613	❌
.server.Server	Fat	92	🚩
.server.Server	WMC	123	🚩
.server.Server.checkWinner()	CC	21	❌
.server.Server.checkWinner()	ELOC	119	🚩
.server.ServerTest.setupAll()	ELOC	88	🚩
.server.ServerTest.testFlow()	ELOC	97	🚩

- il warning per la misura della lunghezza [ELOC] dei metodi setupAll() e testFlow() non lo consideriamo poichè riguarda metodi della classe di test ServerTest
 - la misura della complessità ciclomatica [CC] e della lunghezza [ELOC] del metodo checkWinner() della classe Server non rispettavano i valori di soglia: è stato quindi eseguito un refactoring del metodo, separandolo in diversi metodi e rendendolo più chiaro e manutenibile