Immagine che contiene Carattere, cerchio, simbolo, bianco

Descrizione generata automaticamente

Documentazione progetto di

**Testing e Verifica del Software**

cod. corso 21056

Laurea Magistrale in Ingegneria Informatica

Università degli Studi di Bergamo

A.A. 2024/2025

Prof. Angelo Gargantini

Progetto di:

**Davide Gamba**, matr. 1053470

[1. Introduzione e Analisi dei Requisiti 1](#_Toc204543354)

[1.1 – Descrizione del Progetto 1](#_Toc204543355)

[1.2 – Analisi dei requisiti 1](#_Toc204543356)

[1.3 – StateChart UML 5](#_Toc204543357)

[2. ASMETA 6](#_Toc204543358)

[2.1 – Simulazione/animazione del modello 6](#_Toc204543359)

[2.2 – Scenari con Avalla 13](#_Toc204543360)

[2.3 – Model Checking 14](#_Toc204543361)

[3. Implementazione in Java 17](#_Toc204543362)

[3.1 – Implementazione delle funzionalità in Java 17](#_Toc204543363)

[3.2 – Testing del programma con Junit 18](#_Toc204543364)

[3.3 – Analisi statica del Codice 21](#_Toc204543365)

[4 – JML 29](#_Toc204543366)

[4.1 – Definizione dei contratti con JML 29](#_Toc204543367)

[4.2 – Dimostrazione dei contratti con ESC 32](#_Toc204543368)

[5 – Continuos Integration with Github Actions 33](#_Toc204543369)

# 1. Introduzione e Analisi dei Requisiti

## 1.1 – Descrizione del Progetto

**SmartPrinter** è un progetto finalizzato alla modellazione e simulazione delle principali funzionalità di una stampante moderna, in grado di eseguire le operazioni di stampa e scansione di documenti. Il progetto pone particolare attenzione alla gestione delle risorse fisiche (come carta e toner) e alla rilevazione degli errori più comuni, come inceppamenti e guasti.

La stampante integra anche un meccanismo di autenticazione, in modo tale da consentire l’accesso ai servizi solo agli utenti autorizzati, garantendo un uso controllato, sicuro e monitorabile del dispositivo.

L’obiettivo del progetto è fornire una rappresentazione formale del comportamento della stampante nei vari stati operativi, assicurando affidabilità, coerenza e una gestione efficace delle risorse e degli errori.

## 1.2 – Analisi dei requisiti

Una grande azienda multinazionale operante nel settore del Management Consulting intende introdurre una nuova linea di stampanti multifunzione di alta qualità, con l’obiettivo di supportare il personale nella gestione e distribuzione della documentazione aziendale.

L’iniziativa nasce dall’esigenza di standardizzare la produzione di documenti interni, come report e presentazioni utilizzati nelle riunioni tra i dirigenti, nonché la documentazione esterna destinata ai clienti, garantendo uniformità e professionalità nella comunicazione.

L’obiettivo principale è quello di definire processi documentali efficienti, ridurre al minimo gli sprechi di risorse e garantire un utilizzo controllato delle stampanti, limitato esclusivamente al personale autorizzato.

Ai fini della realizzazione del progetto, sono stati individuati i seguenti requisiti funzionali:

**1. Procedura di avvio della stampante:** La stampante deve poter essere accesa mediante un apposito pulsante di accensione (On/Off). Durante la fase di avvio, il sistema esegue una serie di controlli hardware e software per verificare il corretto funzionamento del dispositivo.

* In caso di esito positivo dei controlli, la stampante procede richiedendo l'autenticazione dell'utente.
* In caso di esito negativo, il dispositivo entra in modalità *fuori servizio* e richiede l’intervento di un tecnico qualificato per la diagnosi e la risoluzione del guasto.

**2. Identificazione e autenticazione dell’utente:** Al fine di garantire un utilizzo controllato e monitorato del dispositivo, è previsto un meccanismo di autenticazione che consenta l’accesso esclusivamente agli utenti autorizzati.  
Nel dettaglio, all’utente viene richiesto di presentare il badge aziendale presso l’apposito lettore, seguito dall’inserimento del codice PIN personale.

* Se il tesserino presentato non risulta autorizzato, la procedura viene interrotta e all’utente viene richiesto di ripetere l’operazione con un badge valido.
* Se il tesserino è valido, il processo prosegue con l’inserimento del PIN:
  + Se il PIN è corretto, l’utente può procedere ad utilizzare la stampante.
  + In caso di PIN errato, la procedura viene annullata e l’utente dovrà ripetere l’intero processo di autenticazione, a partire dalla presentazione del tesserino.

**3. Funzionalità disponibili:** Una volta completata con successo la procedura di autenticazione, l’utente viene associato alla sessione in corso della stampante.

Le operazioni disponibili sono le seguenti:

* ***Stampa in Bianco e Nero***
* ***Stampa a Colori***
* ***Scansione di Documenti***

Considerata la finalità principale della nuova linea di stampanti, ovvero supportare la standardizzazione della produzione di report e documentazione aziendale, *ciascuna operazione di stampa prevede* ***l’utilizzo esatto di 10 fogli***, corrispondenti alla lunghezza media di un documento aziendale.

Poiché si tratta di dispositivi progettati per stampe ad alta qualità, in fase di progettazione è stato stimato che *ogni operazione di stampa ha un impatto del* ***5% sulla capacità delle cartucce di toner***. In particolare:

* Una stampa in bianco e nero riduce il livello del **toner nero** del 5%.
* Una stampa a colori riduce il livello di **entrambi i toner (nero e colore)** del 5%.

Per quanto riguarda la scansione, *l’operazione può essere effettuata solo se è stato collegato almeno un dispositivo di ricezione* alla stampante. Il collegamento può avvenire tramite **connessione wireless** oppure tramite **cavo fisico** (USB Type-C).

Infine, per garantire e promuovere un uso consapevole delle risorse aziendali e limitare gli sprechi di carta, *ogni utente dispone di un credito virtuale mensile*, inizialmente pari a **1000 crediti** ed *è previsto un costo di* ***50 crediti per ogni stampa*** (sia in bianco e nero che a colori), mentre l’operazione di scansione non ha un costo.

**4. Stampante in uso e gestione degli errori di stampa:** Una volta selezionata dall’utente l’operazione desiderata, si possono verificare i seguenti casi:

* *Stampa in Bianco e Nero*: Se l’utente ha abbastanza credito ed i valori di toner nero e carta sono sufficienti, la stampante inizia ad eseguire il task, in caso contrario verrà notificato quale è il problema tramite un messaggio sul display. La stampa in bianco e nero richiede circa 2 secondi.
* *Stampa a colori*: Se l’utente ha abbastanza credito ed i valori di toner nero, toner a colori e carta sono sufficienti, la stampante inizia ad eseguire il task, in caso contrario verrà notificato quale è il problema tramite un messaggio sul display. La stampa in bianco e nero richiede circa 3 secondi
* *Scansione*: La scansione di un documento viene effettuata se è stato collegato almeno un dispositivo per la ricezione, altrimenti viene mostrato un messaggio. La stampa in bianco e nero richiede circa 4 secondi.

Se l’operazione di stampa/scansione si conclude correttamente, la stampante torna nello stato operativo, consentendo all’utente di effettuare ulteriori operazioni o, in alternativa, di terminare la propria sessione spegnendo il dispositivo tramite il tasto On/Off.

Qualora, invece, la carta non fosse stata inserita correttamente nell’apposito vano, durante l’operazione di stampa potrebbe verificarsi un inceppamento. In tal caso, la macchina interrompe l’operazione e notifica l’errore all’utente, il quale dovrà sistemare la carta e ripetere l’operazione desiderata. In caso di errore, l’importo relativo alla stampa non viene detratto dal credito dell’utente.

## 1.3 – StateChart UML

Prima di passare alla fase di implementazione, è stata realizzata una State Machine con *StarUML* per rappresentare in modo chiaro gli stati del sistema e le relative transizioni, con il fine di facilitare la comprensione del comportamento dinamico della stampante.

Immagine che contiene diagramma, Piano, Disegno tecnico, schematico

Il contenuto generato dall'IA potrebbe non essere corretto.

# 2. ASMETA

Per rappresentare formalmente il comportamento della stampante e verificarne la correttezza, è stato sviluppato un modello in ***AsmetaL***, il linguaggio formale utilizzato dall’ambiente **ASMETA** per la definizione di Abstract State Machines (ASM).

Il modello consente di descrivere con precisione gli stati del sistema, le transizioni tra essi e le regole di comportamento in funzione delle azioni dell’utente e delle condizioni operative, fornendo una base solida per l’analisi e la validazione del software prima dell’implementazione concreta.

## 2.1 – Simulazione/animazione del modello

Una volta sviluppato il modello, è stata effettuata una simulazione tramite lo strumento ***AsmetaS***, che consente di animare le regole definite e osservare l’evoluzione degli stati nel tempo.

Di seguito vengono riportati alcuni screenshot rappresentativi delle principali fasi della simulazione, accompagnati da una breve descrizione.

***1 – Accensione della stampante***

Immagine che contiene schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

***2 – Avvio della stampante con controllo Guasta/Non Guasta***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***3 – Stampante va fuori servizio e viene riparata***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***4 – Esibizione del Badge***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***5 – Inserimento Pin utente***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***6 – Scelta di un’azione***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***7 – Stampa a colori terminata con successo***

**Immagine che contiene schermata

Il contenuto generato dall'IA potrebbe non essere corretto.**

***8 – Scansione terminata con successo***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***9 – Stampa in bianco e nero terminata con errore***

**Immagine che contiene schermata, testo, design

Il contenuto generato dall'IA potrebbe non essere corretto.**

***10 – Carta inceppata sistemata***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***11 – Spegnimento della stampante***

**Immagine che contiene schermata, testo

Il contenuto generato dall'IA potrebbe non essere corretto.**

***12 – Autenticazione di un nuovo utente, che sbaglia l’inserimento del Pin***

**Immagine che contiene schermata

Il contenuto generato dall'IA potrebbe non essere corretto.**

***13 – L’utente ripete il processo di autenticazione ed effettua una Stampa in bianco e nero***

**Immagine che contiene schermata

Il contenuto generato dall'IA potrebbe non essere corretto.**

## 2.2 – Scenari con Avalla

Al fine di garantire che il sistema reagisca in modo coerente rispetto ai requisiti stabiliti, sono stati definiti ed eseguiti diversi **scenari di utilizzo** nel linguaggio *Avalla*, tali scenari sono stati progettati per coprire i casi d’uso principali, comprese situazioni corrette e situazioni di errore.

In particolare, gli scenari definiti sono stati validati con Vc per verificare la copertura e sono state utilizzate le seguenti primitive Avalla:

* **Set**: per impostare il valore delle monitorate a un valore specifico, simulando cosi l’ambiente.
* **Check**: verificare il valore delle funzioni controllate dello stato corrente.
* **Step:** Per eseguire un passo dell’ASM.
* **Invariant:** Per verificare che una proprietà è sempre vera durante

l’esecuzione dello scenario

* **Exec:** Per esegue una regola di transizione, tipicamente una regola per impostare ad un certo valore le variabili controllate all’inizio dell’esecuzione dello scenario.

Gli scenari definiti e validati sono:

1. *Un utente si collega ed effettua una stampa in bianco e nero, poi si scollega e si collega un altro utente che effettua una stampa a colori.*
2. *Un utente vuole ad accedere ma la stampante è guasta, dopo due stati viene sistemata e l'utente effettua l'accesso. L'utente esegue poi due scansioni, una per ogni dispositivo (tramite wireless e cavo).*
3. *Un utente accede ed effettua una stampa ma la carta si inceppa, l'utente sistema la carta e ripete la stampa, l'utente si scollega e poi si collega un altro utente che effettua una scansione.*
4. *Un utente inserisce il pin sbagliato e ripete quindi il processo di autenticazione.*
5. *La scansione non viene eseguita perché non è stato collegato nessun dispositivo.*
6. *La stampa non viene consentita con Toner nero insufficiente.*
7. *La stampa a colori non viene consentita con toner a colori insufficiente.*
8. *La stampa non viene consentita con quantità fogli insufficiente.*
9. *La stampa non viene consentita con credito utente insufficiente.*

## 2.3 – Model Checking

Successivamente, con l’obiettivo di verificare la correttezza dell’implementazione Asmeta rispetto a tutte le possibili evoluzioni temporali del sistema, è stata condotta un’attività di Model Checking utilizzando il model checker ***NuSMV***. In particolare, è stato definito un insieme di proprietà espresse in **CTL (Computation Tree Logic),** la cui validità è stata verificata sul modello.

Le proprietà temporali sono state verificate su una versione semplificata del modello Asmeta, in cui è stata rimossa la componente relativa all’autenticazione dell’utente, mantenendo esclusivamente la parte operativa della stampante.

***StateChart del modello semplificato***

*Immagine che contiene diagramma, Piano, linea, Disegno tecnico

Il contenuto generato dall'IA potrebbe non essere corretto.*

***Le proprietà CTL verificate sono:***

**P1: è sempre possibile che esista uno stato futuro in cui se**

**la stampante è pronta,nello stato successivo potrebbe essere in Uso**

CTLSPEC ag(ef(printerState = PRONTA implies ex(printerState = INUSO)))

**P2: è sempre possibile che si verifichi uno stato futuro in cui se la macchina è fuori servizio rimane fuori servizio**

CTLSPEC ag(ef(printerState = OUTOFSERVICE implies eg(printerState = OUTOFSERVICE)))

**P3: Esiste uno stato futuro in cui il Toner nero è terminato mentre il Toner a colori no**

CTLSPEC ef(tonerNero = 0 and tonerColore > 0)

**P4: Esiste uno stato futuro in cui sia il Toner nero che quello a colori sono terminati mentre i fogli no**

CTLSPEC ef(tonerNero = 0 and tonerColore = 0 and fogliCarta > 0)

**P5: Non può esistere uno stato futuro in cui il toner a colori è terminato mentre il toner nero no**

CTLSPEC (not ef(tonerNero > 0 and tonerColore = 0))

**P6: In qualsiasi stato si trovi la macchina, esiste un percorso che la porta nello stato futuro di PRONTA**

CTLSPEC ag(ef(printerState = PRONTA))

**P7: Nella stampante ci saranno sempre almeno 300 fogli**

**(200 fogli per finire il toner -> 20 stampe -> max 200 fogli consumati)**

CTLSPEC ag(fogliCarta >= 300)

**P8: Una volta che il toner è finito, rimane a zero**

CTLSPEC ag(tonerNero = 0 implies ag(tonerNero = 0))

**P9: Una stampa in Bianco e nero non dura più di 2 secondi**

CTLSPEC ag((selectedService = PRINTBN and printerState = INUSO)

implies au(secondi <= 2, (printerState = PRONTA or printerState = ERRORE)))

**P10: Una stampa a colori non dura più di 3 secondi**

CTLSPEC ag((selectedService = PRINTCOL and printerState = INUSO)

implies au(secondi <= 3, (printerState = PRONTA or printerState = ERRORE)))

**P11: Una scansione non dura più di 4 secondi**

CTLSPEC ag((selectedService = SCANSIONE and printerState = INUSO)

implies au(secondi <= 4, printerState = PRONTA))

**P12: Se la stampante è pronta ed il toner è finito, la prossima operazione non può essere una stampa**

CTLSPEC ag((printerState = PRONTA and tonerNero = 0)

implies not ex(printerState = INUSO and (selectedService = PRINTBN or selectedService = PRINTCOL )))

**P13: Se la stampante è pronta e nessun dispositivo è stato collegato,**

**la prossima operazione non può essere una scansione**

CTLSPEC ag((printerState = PRONTA and connectedByWireless = false and connectedByCable = false) implies not ex(printerState = INUSO and selectedService = SCANSIONE))

# 3. Implementazione in Java

Viene di seguito illustrata l’implementazione ed il testing del sistema realizzato utilizzando il linguaggio di programmazione ***Java***.

L’obiettivo principale è quello di tradurre le specifiche formali definite nel modello *Asmeta* in una struttura software funzionante, mantenendo la coerenza con il comportamento descritto.

Durante lo sviluppo del software, un ruolo fondamentale è stato svolto dai **test *JUnit***, il cui utilizzo ha supportato la verifica della correttezza funzionale del programma, garantendo il rispetto dei requisiti attesi e facilitando l’individuazione tempestiva di malfunzionamenti.

Successivamente, la qualità del codice è stata analizzata attraverso l’impiego di strumenti di ***analisi statica***: ***Stan4J***, ***SonarQube*** e ***PMD***, al fine di individuare eventuali errori, vulnerabilità e bad practices. Una volta terminata l’analisi, sono state intraprese diverse azioni correttive per migliorare la qualità complessiva del codice, rendendolo più robusto, leggibile e manutenibile.

## 3.1 – Implementazione delle funzionalità in Java

L’implementazione delle funzionalità della stampante è stata quindi realizzata in Java, sfruttando l’IDE *Eclipse* e adottando un approccio modulare e orientato agli oggetti.

La classe centrale del progetto è *SmartPrinter.java*, che rappresenta il cuore del sistema e contiene gli attributi e i metodi necessari per realizzare il comportamento della stampante: gestione dello stato (pronta, in uso, fuori servizio…), dei materiali di consumo (toner, fogli), dei servizi disponibili (stampa bianco e nero, stampa a colori, scansione), nonché la gestione dei guasti delle situazioni di errore durante la stampa.

La classe *Utente.java* è invece responsabile della rappresentazione degli utenti autorizzati all’utilizzo della stampante. Ogni oggetto Utente contiene le informazioni necessarie all'autenticazione e all’utilizzo della stampante, ovvero il proprio numero di badge, il Pin ed il credito associato.

Infine, è stata implementata la classe *InterazioneStampante.java*, che funge da punto d’ingresso per l’utilizzo interattivo del sistema. Questa classe ingloba al suo interno un’istanza di SmartPrinter e fornisce un’interfaccia testuale per l’interazione con la stampante, permettendo così di effettuare il login, scegliere un servizio e visualizzare lo stato del sistema in tempo reale.

## 3.2 – Testing del programma con Junit

Durante lo sviluppo del programma, i test realizzati con il framework ***JUnit*** hanno avuto un ruolo fondamentale nel garantire la correttezza e l’affidabilità del sistema. Grazie ai test è stato possibile **individuare rapidamente comportamenti inattesi**, intervenire in modo tempestivo e migliorare sia la **manutenibilità** che la **robustezza** del codice.

In particolare, per la classe *SmartPrinter.java* è stato adottato un approccio ***Test-Driven Development (TDD)*** per guidare lo sviluppo delle funzionalità più critiche. I test hanno consentito di verificare in modo sistematico il comportamento della stampante in tutti i possibili **stati interni**, assicurando la coerenza con le specifiche del modello ASM.

Inoltre, i metodi responsabili dell’erogazione dei servizi (stampe e scansione) sono stati testati con dei test parametrici per garantire una c***opertura MCDC (Modified Condition/Decision Coverage)***.

Le classi di Test realizzate sono:

1. **SmartPrinterTest1Login.java**: Test Suite che copre tutto il processo di autenticazione e lo stato di guasto.
2. **SmartPrinterTest2StampaBNMCDC.java**: Test Suite parametrica MCDC per la stampa in bianco e nero.
3. **SmartPrinterTest3StampaCOLMCDC.java**: Test Suite parametrica MCDC per la stampa a colori.
4. **SmartPrinterTest4ScansioneMCDC.java**: Test Suite parametrica MCDC per la scansione.
5. **SmartPrinterTest5InUsoErrore.java**: Test Suite per il testing degli stati “In Uso” ed “Errore”.

Inoltre, utilizzando il tool ***CodeCover***, è stato possibile misurare con precisione la copertura del codice, in particolare, eseguendo l’intera suite di test, la copertura degli **statement per la classe *SmartPrinter.java*** risulta essere di circa il **90%**.

***Copertura statement***

**Immagine che contiene schermata, testo, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.**

***Stampa in Bianco e Nero MCDC***

**Immagine che contiene schermata, testo, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.**

***Stampa a colori MCDC***

**Immagine che contiene schermata, testo, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.**

***Scansione MCDC***

**Immagine che contiene schermata, testo, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.**

## 3.3 – Analisi statica del Codice

Successivamente, si è proseguito nell’applicare una serie di tecniche di analisi statica per valutare la qualità del codice, utilizzando i seguenti strumenti.

### 3.3.1 – SonarQube

*SonarQube* è un tool per l'analisi statica del codice, utilizzato per individuare bug, vulnerabilità di sicurezza e bad practices all'interno di un progetto software, permettendo così allo sviluppatore di intervenire per migliorare la qualità del proprio lavoro.

Inizialmente l’intero progetto Java è stato sottoposto ad analisi tramite SonarQube ed il tool ha evidenziato molti warning, possibili miglioramenti del codice e diverse potenziali problematiche.

Immagine che contiene testo, software, Pagina Web, Sito Web

Il contenuto generato dall'IA potrebbe non essere corretto.

A seguito dell’analisi condotta, sono state apportate diverse modifiche correttive per risolvere i problemi segnalati dal tool. In particolare:

1. È stato introdotto un logger dedicato in sostituzione dell’uso di System.out.println().
2. Sono stati rimossi tutti gli import inutilizzati.
3. Sono stati eliminati return booleani ridondanti, sostituendoli con espressioni più concise.
4. È stato aggiunto il blocco default nei costrutti switch in cui mancava.
5. Stringhe duplicate nelle stampe sono state sostituite da costanti static final.
6. Un costrutto switch è stato rimpiazzato con un if per rendere il flusso di controllo più chiaro.
7. Sono stati migliorati i metodi dedicati alla ricerca degli utenti, rendendoli più snelli e leggibili.

Di seguito viene mostrato come, su segnalazione di SonarQube, siano stati migliorati i metodi dedicati alla ricerca degli utenti. ***Si riportano le versioni originarie dei metodi prima della revisione:***

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

***I metodi dopo la correzione:***

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

Si osserva come, a seguito della correzione realizzata successivamente alla segnalazione di SonarQube, sia stato possibile unificare la logica di ricerca in un unico metodo di supporto, eliminando la ridondanza dei due metodi iniziali. Inoltre, la gestione del caso in cui l'utente risulti *null* è ora più concisa ed efficiente.

Infine, analizzando nuovamente il progetto dopo le correzioni si ottengono i seguenti risultati:

Immagine che contiene testo, schermata, software, Pagina Web

Il contenuto generato dall'IA potrebbe non essere corretto.

Dove ora gli unici avvisi segnalati riguardano la lunghezza e la complessità cognitiva del metodo *UsoStampante(),* ovvero il metodo che si occupa di gestire l’uso interattivo della stampante.

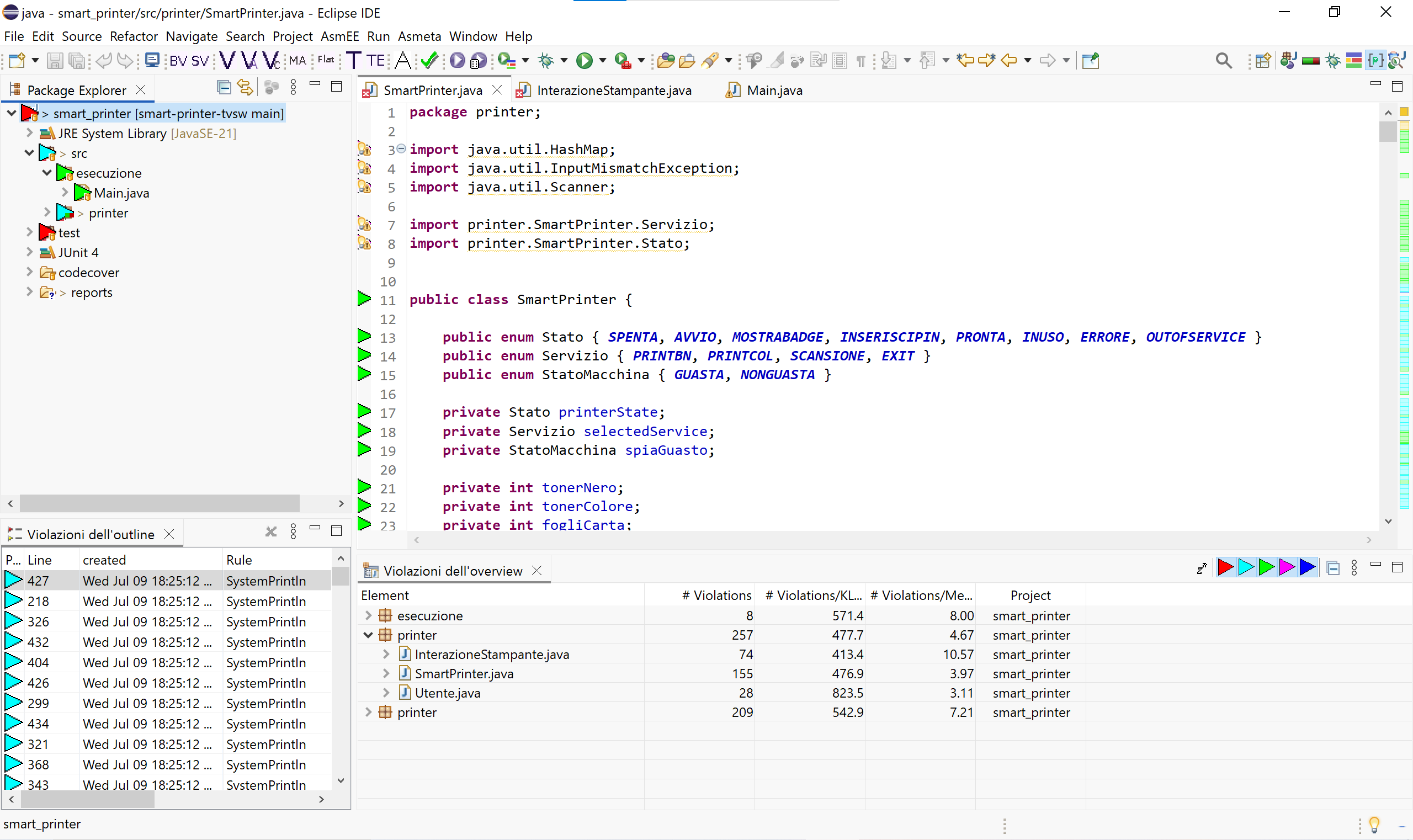
### 3.3.2 – PMD

Anche *PMD* è un tool di analisi statica, ampiamente utilizzato per identificare automaticamente potenziali problemi nel codice, come pratiche di programmazione errate, duplicazioni di codice, variabili inutilizzate, strutture di controllo ridondanti e mancanza di documentazione.

Analogamente a SonarQube, consente di migliorare la qualità del software fin dalle prime fasi dello sviluppo, fornendo suggerimenti concreti su come ottimizzare il codice.

Analogamente a quanto fatto con SonarQube, *PMD* è stato impiegato in due fasi distinte: una prima analisi è stata condotta prima dell’applicazione delle modifiche correttive descritte precedentemente, permettendo di individuare una serie di segnalazioni critiche e non critiche. Successivamente, il tool è stato rieseguito dopo aver effettuato gli interventi di refactoring per valutare l’efficacia degli interventi.

***Prima degli interventi di refactoring***

****

***Dopo gli interventi di refactoring***

**Immagine che contiene testo, schermata, software, linea

Il contenuto generato dall'IA potrebbe non essere corretto.**

Confrontando le due situazioni si può notare come gli interventi di refactoring abbia notevolmente abbassato il numero di warning segnalati da PMD.

### 3.3.3 – Stan4J

Stan4J è un tool di analisi statica specializzato nell’analisi delle dipendenze tra i pacchetti e le classi all'interno di un progetto Java. A differenza di PMD e SonarQube, che si concentrano prevalentemente su problemi legati alla qualità del codice, come la presenza di codice duplicato, bad practices o potenziali bug, Stan4J è progettato per valutare l’architettura complessiva del software, in particolare il rispetto dei principi di modularità e separazione delle responsabilità.

Per quanto riguarda il progetto della stampante, ***Stan4J*** è stato utilizzato per valutare la qualità strutturale del codice per la classe principale *SmartPrinter.java,* concentrandosi sulle **metriche** di complessità, accoppiamento e coesione.

**Analisi Stan4j**

Immagine che contiene schermata, testo, software

Il contenuto generato dall'IA potrebbe non essere corretto.

Analizzando nel dettaglio le metriche:

**Caratteristiche della classe:**

* *Classes*: 3
  + Numero totale di classi utilizzate in *SmartPrinter.java.*
* *Methods*: 38
  + Il numero totale di metodi nella classe (getter, setter e metodi per implementare la logica della stampante).
* *Fields***:** 13
  + Campi della classe *SmartPrinter.java.*
* ***ELOC (Estimated Lines of Code)***: 411
  + **Linee di codice della classe, troppe linee di codice in una classe possono portare ad un problema di manutenibilità.**

**Complessità e metriche di accoppiamento:**

* *Fat*: 120
  + Indica la complessità generale, valori troppo alti indicano che la classe fa “troppe cose”.
* ***Ca (Afferent Coupling)*: 1**
  + Numero di classi che dipendono da *SmartPrinter.java.*
* ***Ce (Efferent Coupling)*: 2**
  + Numero di classi da cui *SmartPrinter.java* dipende.

**Metriche di Chidamber & Kemerer:**

* *WMC (Weighted Methods per Class)*: 90
  + Somma delle complessità ciclomatiche dei metodi. Un numero elevato implica alta complessità logica e difficoltà nel testing.
* *DIT (Depth of Inheritance Tree)*: 1
  + La profondità della gerarchia ereditaria. *SmartPrinter.java* eredita direttamente da Object, senza estendere altre classi.
* ***NOC (Number of Children)***: 0
  + Nessuna classe eredita da *SmartPrinter.java.*
* *CBO (Coupling Between Object classes)*: 4
  + Indica che *SmartPrinter.java* è accoppiata a 4 altre classi.
* ***RFC (Response For a Class)*: 46**
  + Numero di metodi potenzialmente attivati da un utilizzo della classe, sintomo di responsabilità multiple.
* *LCOM (Lack of Cohesion of Methods)*: 164
  + Valore molto alto che segnala **bassa coesione**: i metodi svolgono funzionalità multiple e non lavorano sugli stessi attributi.

**In conclusione:**

Per garantire una stretta aderenza al modello Asmeta, la classe *SmartPrinter.java* è stata sviluppata seguendo un approccio monolitico. Questo criterio ha prodotto a una classe con un elevato numero di linee di codice, incaricata di gestire funzionalità eterogenee e di farsi carico dell’intera logica operativa della stampante. La classe, quindi, concentra in sé un’elevata complessità e responsabilità, con un basso livello di coesione interna.

Una possibile strategia per affrontare questa complessità consisterebbe nell’attuare un’operazione di ***refactoring strutturale,*** suddividendo le diverse funzionalità della stampante in classi distinte e specializzate (es : *Login.java, Autenticazione.java, GestioneServizi.java*) integrate successivamente nella classe *SmartPrinter.java*.

Così facendo si potrebbe ottenere una maggiore modularità del software: classi più snelle con una minore complessità ed una maggiore coesione interna.

Tuttavia, prima di intraprendere un'operazione di refactoring, è fondamentale valutare attentamente costi e benefici. Suddividere il comportamento della stampante in più classi comporterebbe la creazione di ulteriori file, un aumento delle dipendenze e una maggiore complessità architetturale complessiva.

Inoltre, poiché l’attuale implementazione monolitica funziona correttamente ed è già ampiamente testata tramite JUnit, intervenire sul design potrebbe introdurre rischi inutili e compromettere la stabilità del sistema; "*If it ain't broke, don't fix it."*

# 4 – JML

Il ***Design by Contract (DbC)*** è un paradigma di sviluppo software che promuove la scrittura di codice più affidabile e robusto attraverso la definizione esplicita di **contratti** tra componenti. Un contratto specifica ciò che una funzione o un modulo software **si aspetta** (precondizioni), ciò che **garantisce** (postcondizioni) e quali **invarianti** devono essere garantite per tutta l’esecuzione del programma.

In Java, il framework più utilizzato per applicare questo paradigma è ***JML (Java Modeling Language)***, un linguaggio formale che permette di annotare direttamente il codice Java con specifiche verificabili, senza alterarne la semantica. JML consente di esprimere contratti precisi su metodi, classi e attributi, ed è supportato da strumenti che possono verificare automaticamente la coerenza tra implementazione e specifica.

Nel progetto **SmartPrinter**, è stato applicato il Design by Contract modificando e documentando i metodi più importanti e significativi della classe *SmartPrinter.java* con le relative **annotazioni JML**.

## 4.1 – Definizione dei contratti con JML

Nel dettaglio i più importanti metodi modificati e per i quali sono stati scritti i contratti JML sono:

***1 – Metodo aggiungiUtente(Utente user) per la registrazione di un dipendente all’uso della stampante***

**//@ requires numUtenti < utenti.length;**

**//@ ensures this.utenti[\old(numUtenti)] == user;**

**//@ ensures numUtenti == \old(numUtenti) + 1;**

**//@ ensures \result == true;**

**protected boolean aggiungiUtente(Utente user) {**

**utenti[numUtenti] = user;**

**numUtenti++;**

**System.out.println("Utente aggiunto: " + user.getNome() + " : " + user.getBadgeId());**

**return true;**

**}**

***2 – Metodo getUtenteByNumBadge(int numBadge) per recuperare un utente dato il suo badge***

**//@ requires (\forall int i; 0 <= i && i < numUtenti; utenti[i] != null);**

**//@ ensures (\exists int i; 0 <= i && i < numUtenti;utenti[i].getBadgeId() == numBadge) ==> \result != null;**

**//@ ensures (\forall int i; 0 <= i && i < numUtenti;utenti[i].getBadgeId() != numBadge) ==> \result == null;**

**protected Utente getUtentebyNumBadge(int numBadge) {**

**//@ loop\_invariant 0 <= i && i <= numUtenti;**

**//@ loop\_invariant (\forall int j; 0 <= j && j < i; utenti[j].getBadgeId() != numBadge);**

**for(int i = 0; i < numUtenti; i++) {**

**if(utenti[i].getBadgeId() == numBadge)**

**return utenti[i];**

**}**

**return null;**

**}**

***3 – Metodo StampaBN() per la stampa in bianco e nero***

**//@ requires printerState == Stato.PRONTA;**

**//@ requires utenteCorrente.haCreditoSufficiente();**

**//@ requires tonerNero >= 5 && fogliCarta >= 10;**

**//@ ensures printerState == Stato.INUSO;**

**//@ ensures selectedService == Servizio.PRINTBN;**

**//@ ensures tonerNero == \old(tonerNero) - 5;**

**//@ ensures fogliCarta == \old(fogliCarta) - 10;**

**//@ ensures utenteCorrente.getCredito() == \old(utenteCorrente.getCredito())-50;**

**//@ ensures \result == true;**

**protected boolean stampaBN() {**

**printerState = Stato.INUSO;**

**selectedService = Servizio.PRINTBN;**

**tonerNero = tonerNero - 5;**

**fogliCarta = fogliCarta - 10;**

**utenteCorrente.scalaCredito(50);**

**System.out.println("Stampa BN in corso");**

**return true;**

**}**

**4 – Metodo *StampaCOL()* per la stampa a colori**

**//@ requires printerState == Stato.PRONTA;**

**//@ requires utenteCorrente.haCreditoSufficiente();**

**//@ requires tonerNero >= 5 && tonerColore >= 5 && fogliCarta >= 10;**

**//@ ensures printerState == Stato.INUSO;**

**//@ ensures selectedService == Servizio.PRINTCOL;**

**//@ ensures tonerNero == \old(tonerNero) - 5;**

**//@ ensures tonerColore == \old(tonerColore) - 5;**

**//@ ensures fogliCarta == \old(fogliCarta) - 10;**

**//@ ensures utenteCorrente.getCredito() == \old(utenteCorrente.getCredito()) - 50;**

**//@ ensures \result == true;**

**protected boolean stampaCOL() {**

**printerState = Stato.INUSO;**

**selectedService = Servizio.PRINTCOL;**

**tonerNero = tonerNero - 5;**

**tonerColore = tonerColore - 5;**

**fogliCarta = fogliCarta - 10;**

**utenteCorrente.scalaCredito(50);**

**System.out.println("Stampa a Colori in corso");**

**return true;**

**}**

**5 – Metodo *scansione()* per effettuare la scansione**

**//@ requires printerState == Stato.PRONTA;**

**//@ requires collegatoWireless || collegatoCavo;**

**//@ ensures printerState == Stato.INUSO;**

**//@ ensures selectedService == Servizio.SCANSIONE;**

**//@ ensures \result == true;**

**protected boolean scansione() {**

**printerState = Stato.INUSO;**

**selectedService = Servizio.SCANSIONE;**

**System.out.println("Scansione in corso");**

**return true;**

**}**

**6 – Invarianti per la classe *SmartPrinter.java***

**//@ public invariant utenti != null;**

**//@ public invariant numUtenti >= 0 && numUtenti <= utenti.length;**

## 4.2 – Dimostrazione dei contratti con ESC

Per verificare la correttezza dei contratti specificati in JML all'interno della classe *SmartPrinter*.java, è stato utilizzato ESC (Extended Static Checker) tramite il plugin OpenJML integrato in Eclipse. Questo strumento analizza staticamente il codice Java e verifica che le asserzioni JML (come precondizioni, postcondizioni, invarianti di ciclo e classi) vengano rispettate.

Immagine che contiene schermata, software, Software multimediale, Icona del computer

Il contenuto generato dall'IA potrebbe non essere corretto.

La maggior parte dei metodi ha superato la verifica statica con esito positivo. Tuttavia, alcuni metodi sono risultati **non validi**, segnalando la necessità di un’ulteriore revisione delle loro specifiche o della loro implementazione.

# 5 – Continuos Integration with Github Actions

La ***Continuous Integration (CI)*** è una pratica fondamentale nello sviluppo software moderno, che consiste nell'effettuare frequentemente modifiche (anche piccole) al codice in un repository.

Attraverso l’implementazione di una pipeline di Continuous Integration, ogni modifica apportata al codice viene automaticamente sottoposta a una serie di controlli (test) volti a garantire che le nuove modifiche non compromettano il corretto funzionamento del codice già esistente. Questo approccio consente di individuare tempestivamente eventuali errori o regressioni, migliorando l’affidabilità complessiva del software e favorendo uno sviluppo più sicuro, stabile e continuo.

Uno dei più utilizzati strumenti di Continuos Integration sono le ***GitHub Actions***, uno strumento integrato direttamente su GitHub che consente di automatizzare flussi di lavoro come build, test e deployment.

Per il progetto SmartPrinter è stata creata una pipeline che esegue automaticamente i test JUnit definiti in precedenza, garantendo così la stabilità e l’affidabilità del sistema ad ogni aggiornamento del codice.

Nel farlo è stato creato un file *main.yaml* all'interno della cartella. github*/workflows/.* Questo file definisce una pipeline CI suddivisa in due jobs distinti:

* Il job *build* si occupa di compilare il codice sorgente e i casi di test, utilizzando Java 21 e le librerie JUnit necessarie ed infine salva i file .class risultanti come artefatti temporanei.
* Il job *test*, che dipende dal completamento della compilazione, scarica questi artefatti e avvia l’esecuzione dei test JUnit. Ogni test è lanciato in uno step separato, così da individuare facilmente eventuali fallimenti.

La pipeline si attiva automaticamente in tre casi:

1. Al push di modifiche nella cartella del progetto (*code/java/smart\_printer/).*
2. All’apertura di una pull request verso il ramo principale.
3. Manualmente, tramite il pulsante **"**Run workflow" su GitHub grazie alla direttiva workflow\_dispatch.

***1 - Pipeline Continuos Integration***

**Immagine che contiene testo, software, Software multimediale, schermata

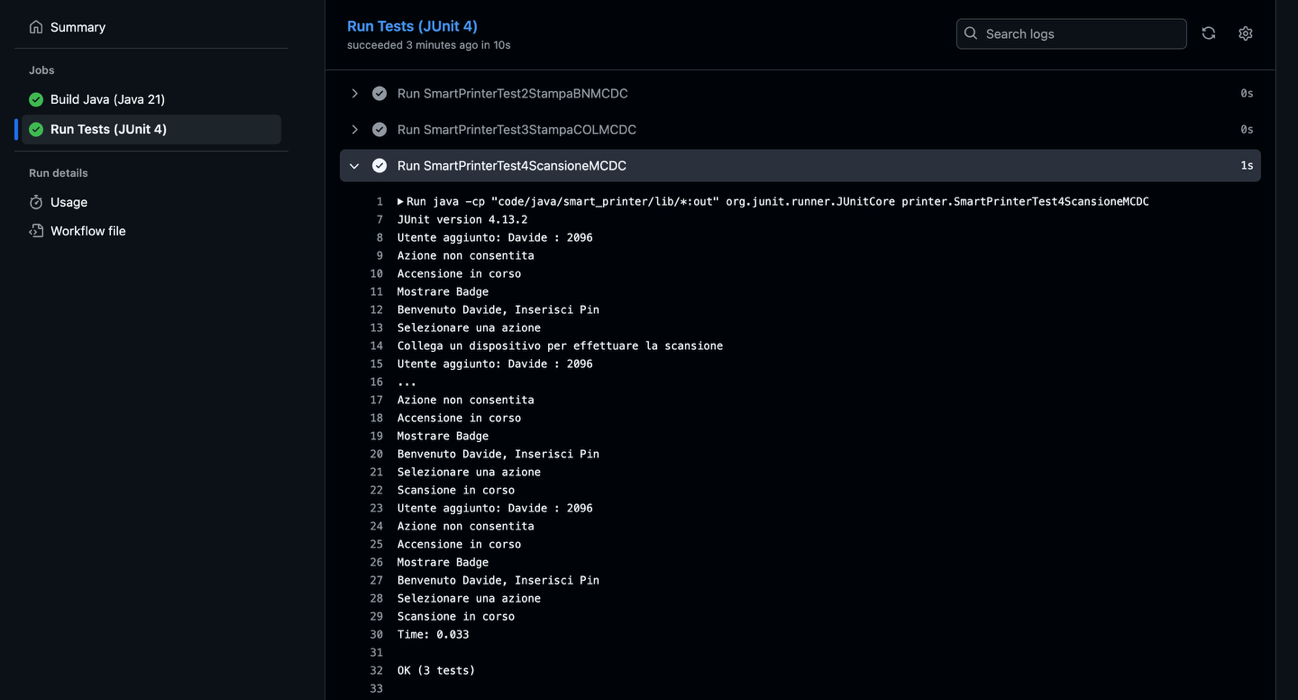
Il contenuto generato dall'IA potrebbe non essere corretto.**

***2 - Step del Job Tests***

**Immagine che contiene software, schermata, testo, Software multimediale

Il contenuto generato dall'IA potrebbe non essere corretto.**

***3 - Output del test ScansioneMCDC***

****