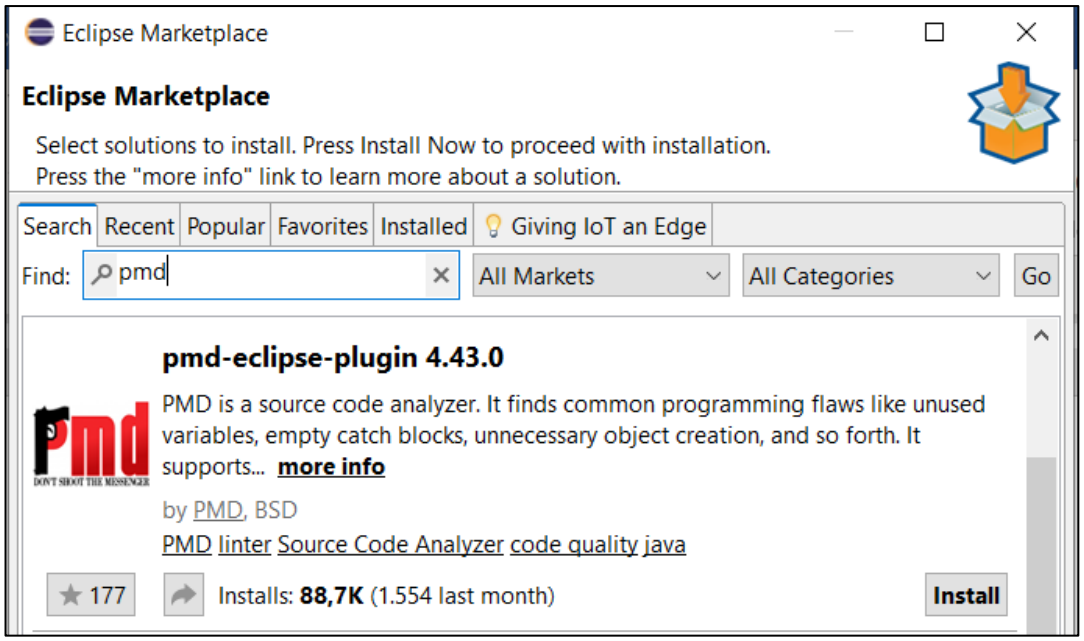
TESTING E VERIFICA DEL SOFTWARE

PMD TUTORIAL

Ing. Piazzi Simona

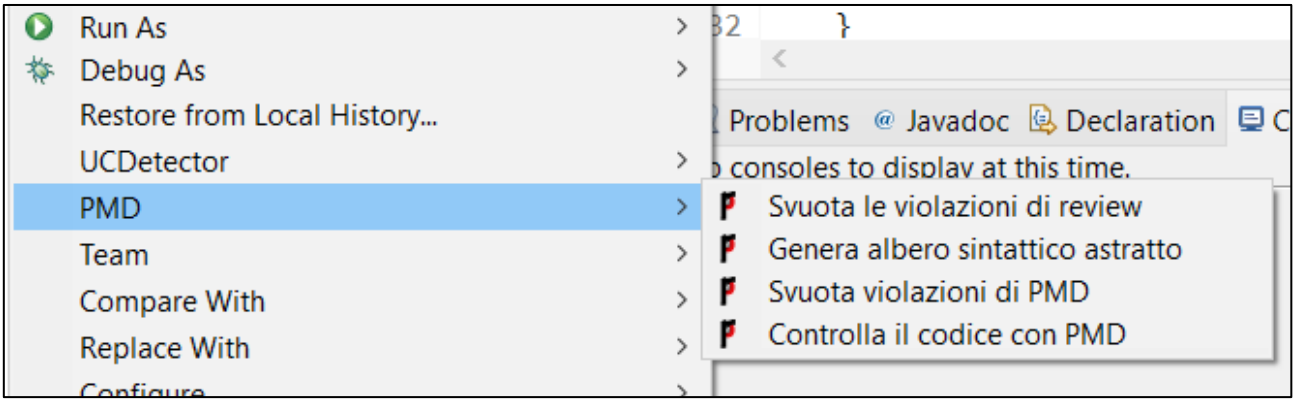
Setup

Per Eclipse è disponibile un plugin per PMD che si può installare tramite Eclipse Marketplace.



Cliccare “Install”, accettare i termini della licenza e cliccare “Finish”.

Una volta terminata l’installazione, riavviare Eclipse. Se l’operazione è andata a buon fine, cliccando con il tasto destro su una qualsiasi classe, fra le diverse opzioni sarà visibile anche quella di PMD.



Per attivare PMD scegliere l’opzione “Controlla il codice con PMD”, mentre per disattivarlo selezionare “Svuota violazioni di PMD”.

Esempio

PMD è un tool per l'analisi statica del codice il cui scopo è quello di rilevare e segnalare problemi e difetti all'interno del codice.

Una volta attivato PMD, verranno quindi mostrati una serie di flag di diversi colori a seconda del livello di violazione rilevato, come mostrato nel codice in figura.

```java
import java.util.List;
import java.util.Scanner;

public class Minigame {
    private int max = 500;
    private int total_attempts = 4;
    private int n;

    public Minigame() {
        n = (int) (Math.random() * max);
    }

    public void play() {
        boolean res = false;
        for(int i=0;i<total_attempts;i++) {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Attempt left: " +(total_attempts-i)+"/"+total_attempts);
            System.out.println("Enter your guess: ");
            int guess = scanner.nextInt();
            if(guess>n) {
                System.out.println("Too high...");
            }
            if(guess == n) {
                System.out.println("Number guessed");
                res = true;
                break;
            }
            if(guess<n) {
                System.out.println("Too low...");
            }
        }
        System.out.println(displayResult(res));
    }

    private String displayResult(boolean b) {
        if(b) {
            return "You win!! The secret number was " + n;
        }else {
            return "You lose... The secret number was " + n;
        }
    }
}
```

Per ciascuna violazione viene fornita la regola di PMD a cui fa riferimento. Consultando la documentazione https://pmd.github.io/latest/pmd_rules_java.html è possibile leggere la descrizione e le possibili soluzioni che possono essere utilizzate per migliorare il codice.

FieldNamingConvention

Categoria Questa violazione appartiene alla categoria "Code Style", ovvero quell'insieme di regole che hanno lo scopo di evidenziare le violazioni delle convenzioni con cui viene scritto il codice.

Problema In questo caso, il problema è dato dalla variabile *total_attempts* (linea 6) che ha un underscore all'interno del suo nome: la convezione prevede infatti che questo tipo di variabili vengano scritte con la notazione Camel case.

Soluzione Per risolvere il problema è sufficiente rinominare la variabile secondo la notazione Camel case: *totalAttempts*.

SystemPrintln

Categoria Questa violazione appartiene alla categoria "Best Practices", ovvero quell'insieme di regole che hanno lo scopo di rafforzare l'uso corretto delle best practices.

Problema In questo caso, il problema è dovuto alla presenza di System.out.println(). Normalmente queste linee sono utilizzate per la fase di debug del codice, per cui è buona pratica sostituirle con un logger che può essere abilitato o disabilitato a seconda del caso.

Soluzione Per risolvere il problema è necessario inserire un logger e sostituirlo alle linee di System.out.println(), indicando eventualmente anche il livello di log.

CommentRequired

Categoria Questa violazione appartiene alla categoria "Documentation", ovvero quell'insieme di regole che hanno lo scopo di controllare che la documentazione essenziale sia presente.

Problema In questo caso, il problema è dovuto alla mancanza di documentazione javadoc a livello di classe, di campi e di metodi pubblici.

Soluzione Per risolvere il problema è necessario aggiungere la documentazione dove richiesta.

NoPackage

Categoria Code Style

Problema In questo caso, il problema è dovuto al fatto che la classe Minigame non ha un package proprio ma si trova in quello di default. Normalmente ogni classe dovrebbe essere inserita in uno specifico package per una migliore organizzazione.

Soluzione Per risolvere il problema è sufficiente spostare la classe Minigame dal package di default ad uno specifico.

ImmutableField

Categoria Questa violazione appartiene alla categoria “Design”, ovvero quell’insieme di regole che segnalano problemi legati al design del programma.

Problema In questo caso, il problema è dato dal fatto che i tre campi della classe Minigame, una volta inizializzati, non cambiano più il proprio valore all’interno del codice.

Soluzione Per risolvere il problema è sufficiente aggiungere la keyword *final*.

FinalFieldCouldBeStatic

Categoria Design

Problema In questo caso, i campi *max* e *totalAttempts* possono essere resi delle costanti.

Soluzione Per risolvere il problema è sufficiente rendere i due campi costanti.

ShortVariable

Categoria Code Style

Problema Il campo *n* ha un nome troppo corto e poco esplicativo. Il nome di una variabile dovrebbe far si che sia chiaro ciò per cui viene utilizzata.

Soluzione Per risolvere il problema è sufficiente cambiare il nome del campo in *numberToGuess*.

AvoidInstantiatingObjectsInLoops

Categoria Questa violazione appartiene alla categoria “Performance”, ovvero quell’insieme di regole che segnalano problemi legati a costrutti che, se ottimizzati, possono portare il programma ad avere performance migliori.

Problema Il problema in questo caso è dato dalla variabile *scanner* che viene inizializzata all’interno del ciclo for. Questa operazione è piuttosto dispendiosa per cui, se possibile, è meglio dichiarare le variabili all’esterno dei cicli.

Soluzione Per risolvere il problema è sufficiente spostare la linea 16 all’esterno del ciclo for.

CloseResource

Categoria Questa violazione appartiene alla categoria “Error Prone”, ovvero quell’insieme di regole che segnalano problemi legati a costrutti caotici o inclini a provocare degli errori a runtime.

Problema Il problema in questo caso è dato dalla variabile *scanner* che non viene mai chiusa dopo essere stata utilizzata.

Soluzione Per risolvere il problema è sufficiente inserire l’istruzione per chiudere la risorsa dopo il ciclo for.

OnlyOneReturn

Categoria Code Style

Problema Di norma un metodo dovrebbe avere un solo punto di uscita che coincide con l'ultima istruzione riportata nel metodo stesso. Il problema in questo caso è quindi dato dalla presenza di un doppio return nella funzione displayResult().

Soluzione Per risolvere il problema è sufficiente sistemare il codice in modo che abbia un unico return.

UnnecessaryImport

Categoria Code Style

Problema Sono presenti alcuni import che possono essere rimossi.

Soluzione Rimuovere gli import non utilizzati (Ctrl + O).

Di seguito si riporta il codice con le diverse correzioni.

```java
package minigames;

import java.util.Scanner;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/**
 * Minigame to guess a number
 */
public class Minigame {
    /**
     * Upperbound for the number range
     */
    private final static int MAX = 500;
    /**
     * Number of available attempts
     */
    private final static int TOTAL_ATTEMPTS = 4;
    /**
     * Number to guess
     */
    private final int numberToGuess;

    /**
     * Logger for debugging purposes
     */
    private static final Logger LOGGER = LogManager.getLogger(Minigame.class.getName());

    /**
     * Random number generation
     */
    public Minigame() {
        numberToGuess = (int) (Math.random() * MAX);
    }
```

```java
    /**
     * Play the minigame
     */
    public void play() {
        boolean res = false;
        final Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < TOTAL_ATTEMPTS; i++) {
            LOGGER.debug("Attempt left: {}/{}", TOTAL_ATTEMPTS - i, TOTAL_ATTEMPTS);
            LOGGER.debug("Enter your guess: ");
            final int guess = scanner.nextInt();
            if (guess > numberToGuess) {
                LOGGER.debug("Too high...");
            }
            if (guess == numberToGuess) {
                LOGGER.debug("Number guessed");
                res = true;
                break;
            }
            if (guess < numberToGuess) {
                LOGGER.debug("Too low...");
            }
        }

        if (res) {
            LOGGER.debug("{}", () -> displayResult(true));
        }else {
            LOGGER.debug("{}", () -> displayResult(false));
        }

        scanner.close();
    }

    private String displayResult(final boolean result) {
        return result ? "You win!! The secret number was " + numberToGuess:
            "You lose... The secret number was " + numberToGuess;
    }
```