Texto

Descripción generada automáticamente

LAB REPORT FOR CODE INSPECTION

SOFTWARE ENGINEERING II

ALEX VILLEGAS

PAO II 2023

Tabla de contenido

[Introduction 3](#_Toc150104491)

[Workshop 3](#_Toc150104492)

[Development 3](#_Toc150104493)

[Challenge 20](#_Toc150104494)

[Development 20](#_Toc150104495)

[Conclusion 46](#_Toc150104496)

[Recommendations 46](#_Toc150104497)

[References 47](#_Toc150104498)

[Resources 47](#_Toc150104499)

# Introduction

In this workshop, we are going to talk about the Code Inspection in software engineering and show a practical case in which we are using the PMD tool for the code inspection.

Code testing is a static test designed to review software code and check for errors. By simplifying all initial defect detection processes, it helps to reduce the rate of defect propagation and avoid later detection of defects. Review of this code is part of the review of any application [1].

Code Inspection is made for:

1. Check for any error that is present in the software code [1].
2. Identify any required process improvement [1].
3. Check whether the coding standard is followed or not [1].
4. Involving peer examination of codes [1].
5. Documenting the defects in the software code [1].

PMD is a source code analyzer. It can find common programming errors like unused variables, empty catch blocks, unnecessary object creation, etc. It supports Java, JavaScript, Salesforce.com Apex and Visualforce, PLSQL, Apache Velocity, XML, XSL. Additionally, it includes CPD, the copy-paste-detector. CPD finds duplicated code in Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, MATLAB, Python, Go, Swift and Salesforce.com Apex and Visualforce [2].

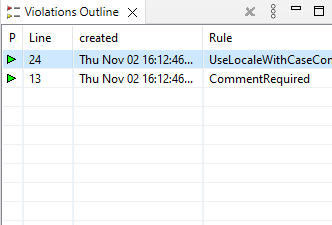
The primary objectives for this workshop are to understand the importance of code quality, identify common coding issues, and learn how automated tools like PMD can assist in code inspection.

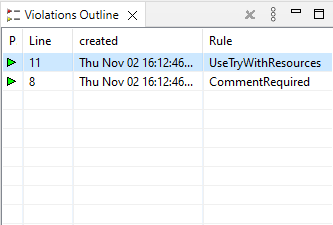
This workshop has two parts: one designed for this workshop and the other one that is a challenge that involves using the previous workshop about Coding Standards.

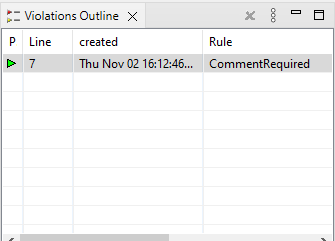
# Workshop

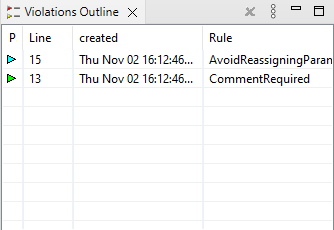
## Development

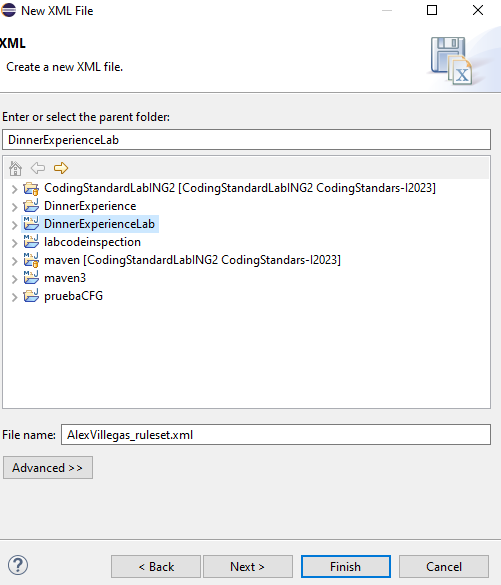
Interfaz de usuario gráfica, Aplicación, Tabla, Excel

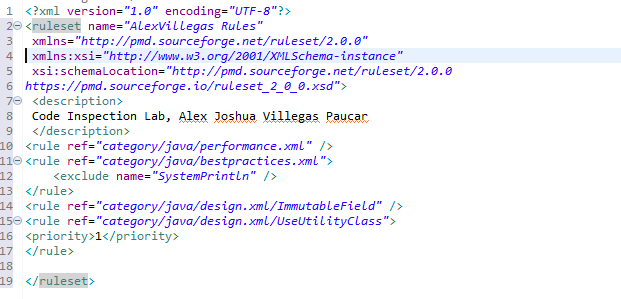
Descripción generada automáticamente





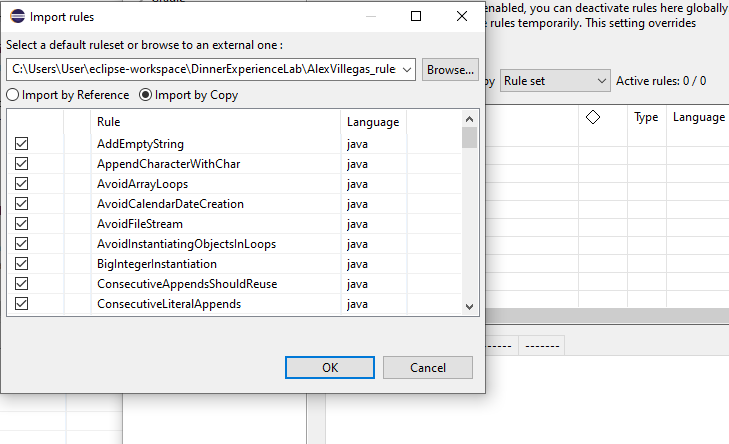


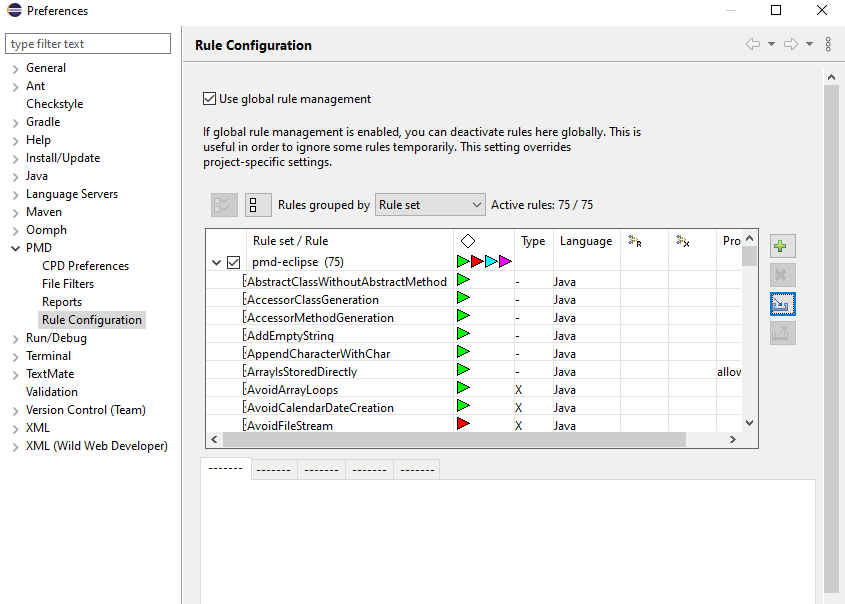


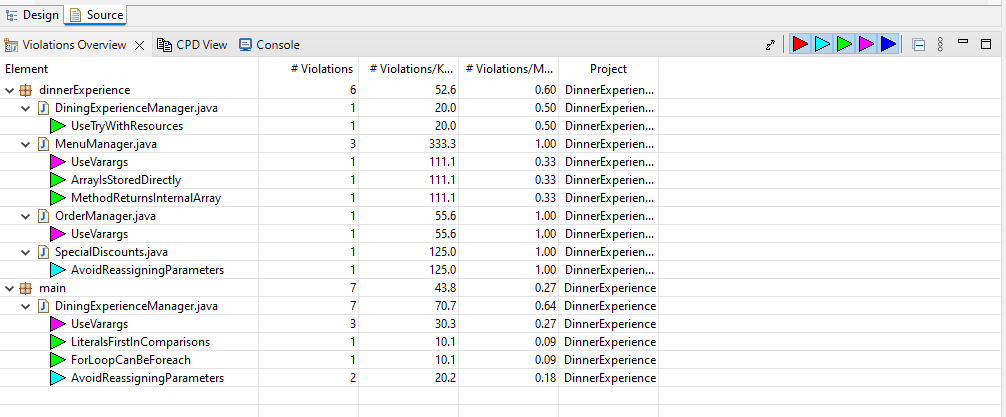


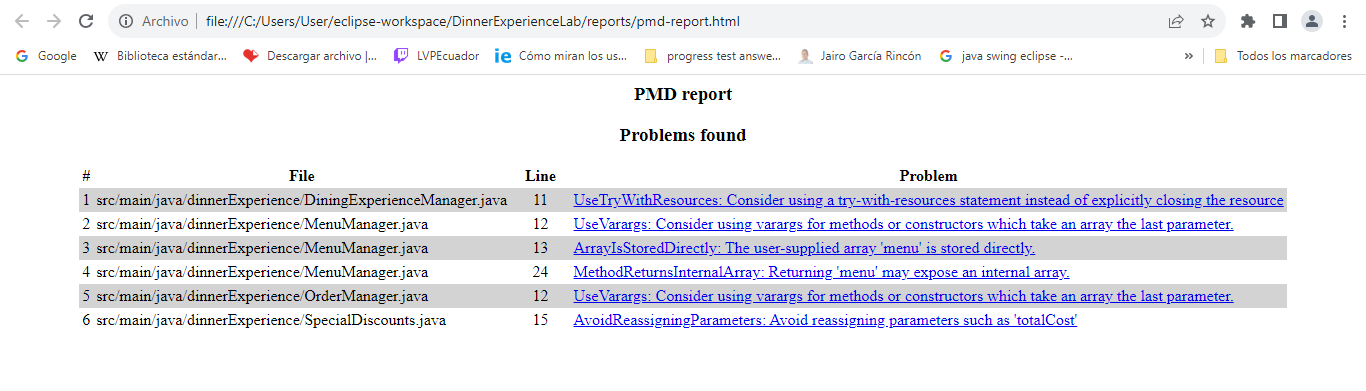
Interfaz de usuario gráfica

Descripción generada automáticamente

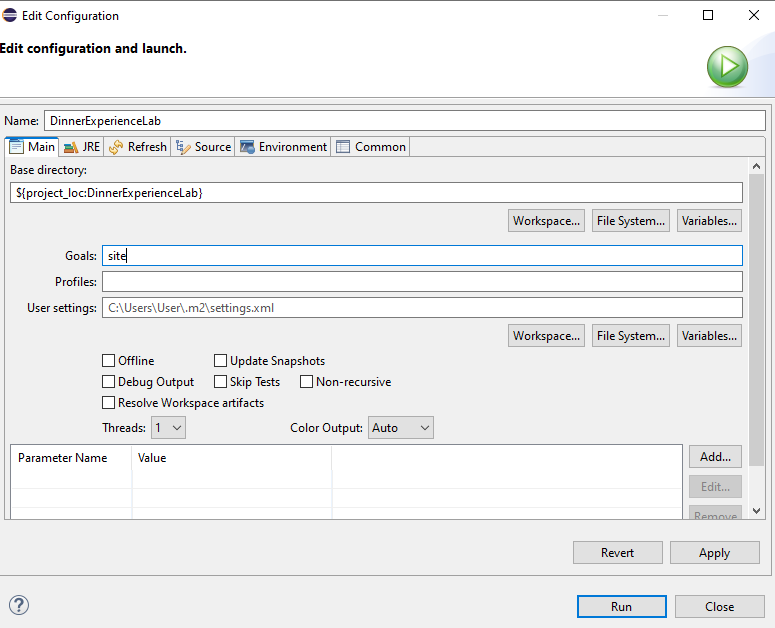


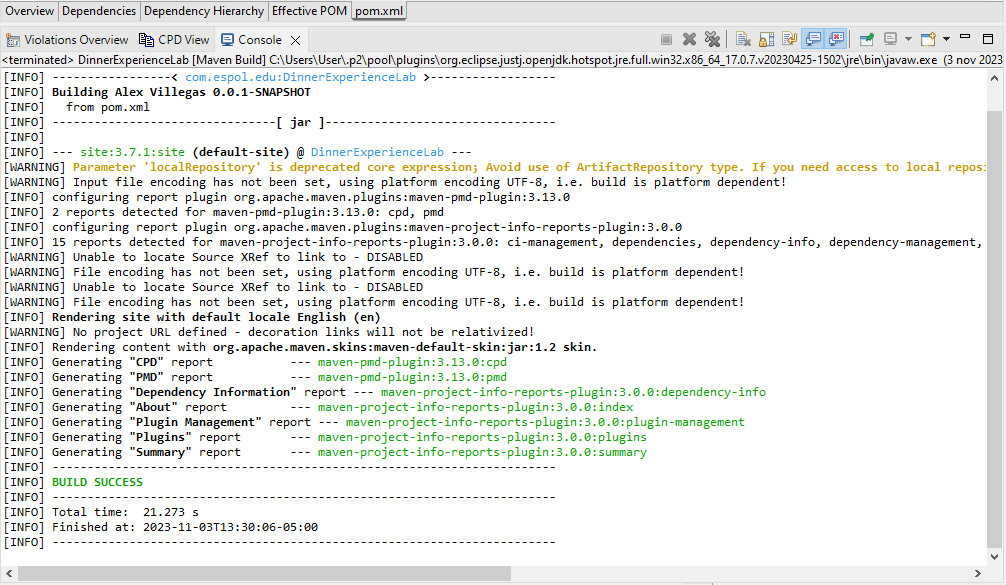
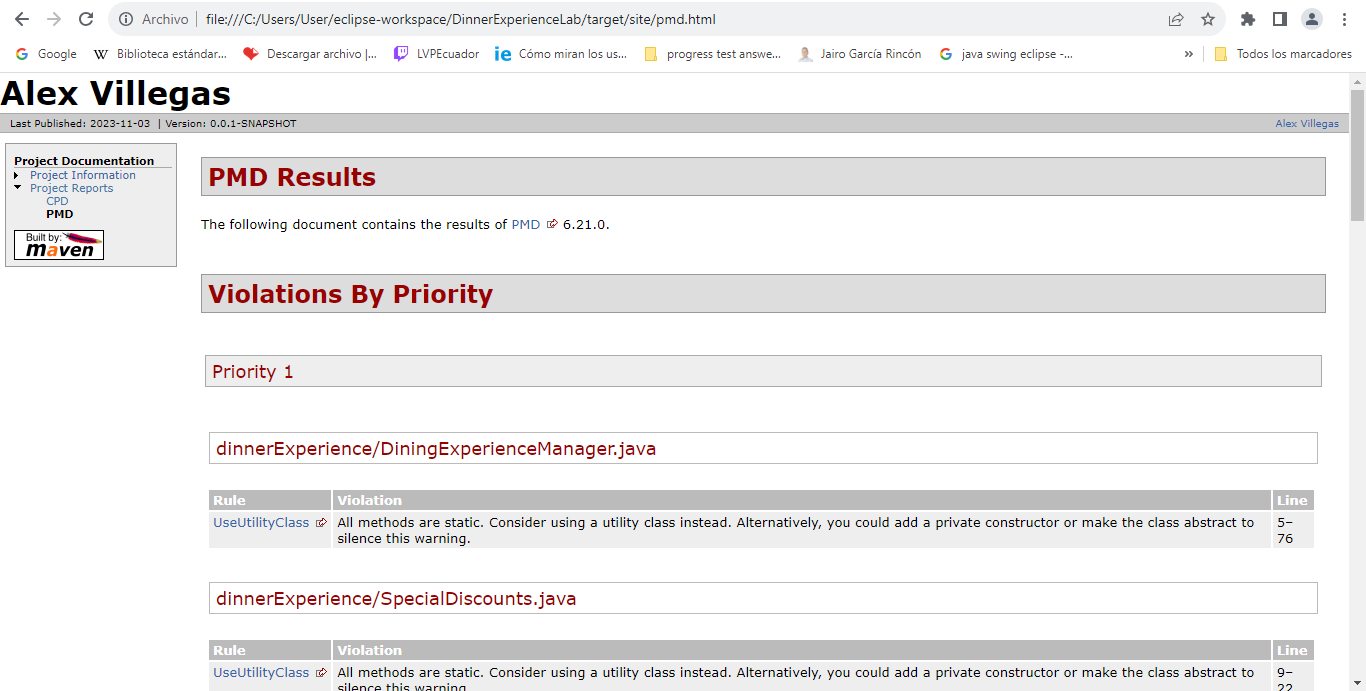


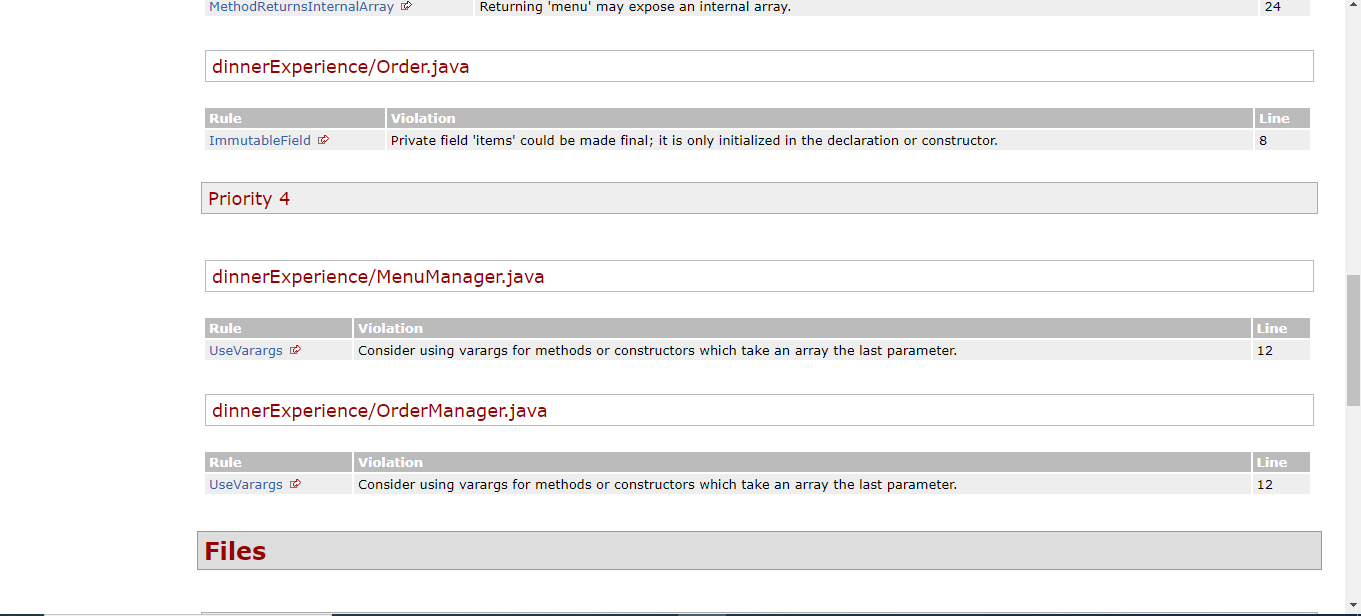
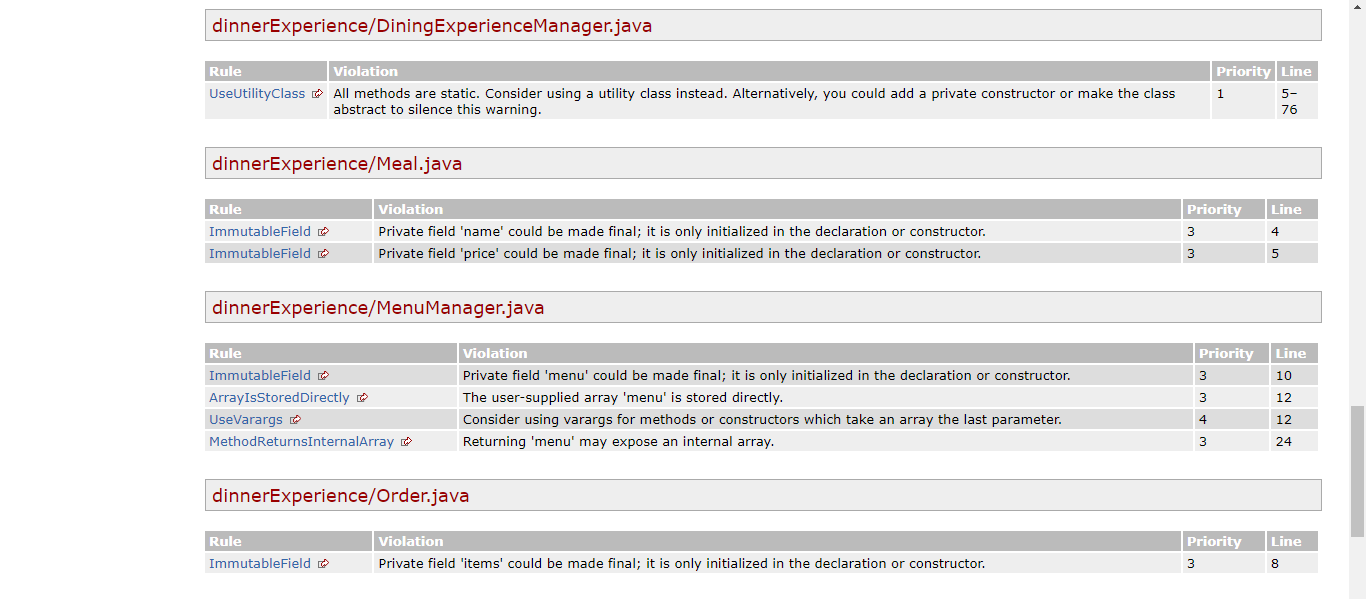
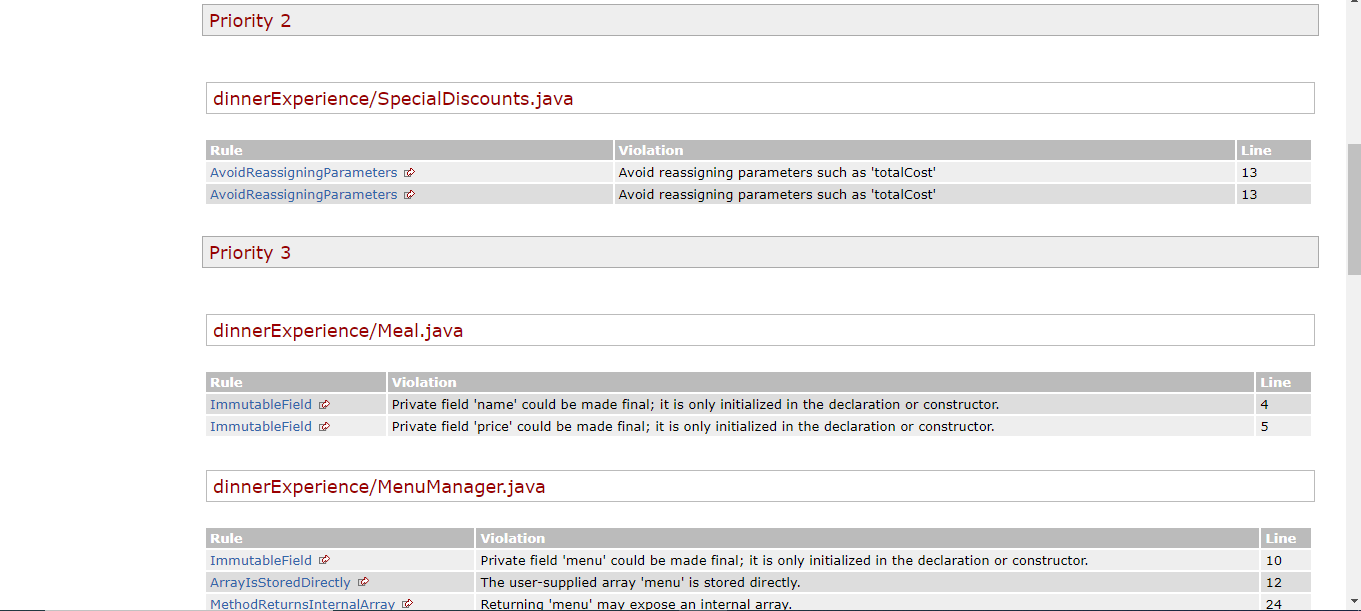




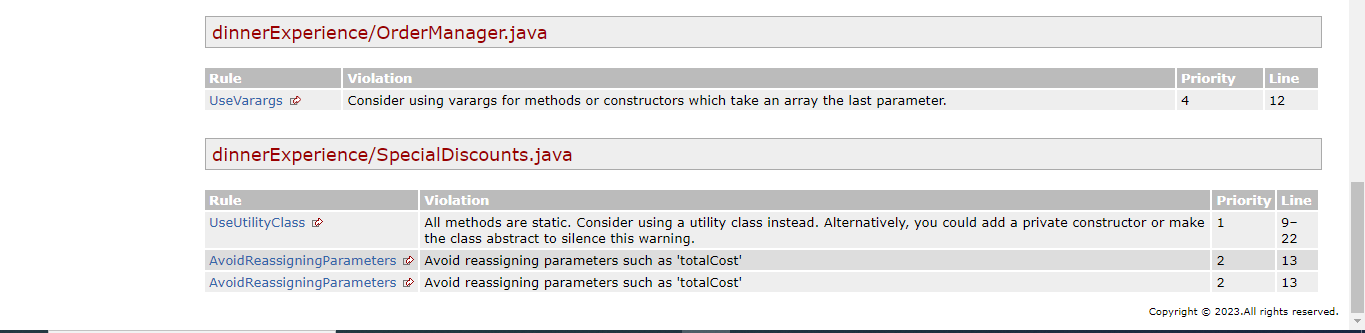
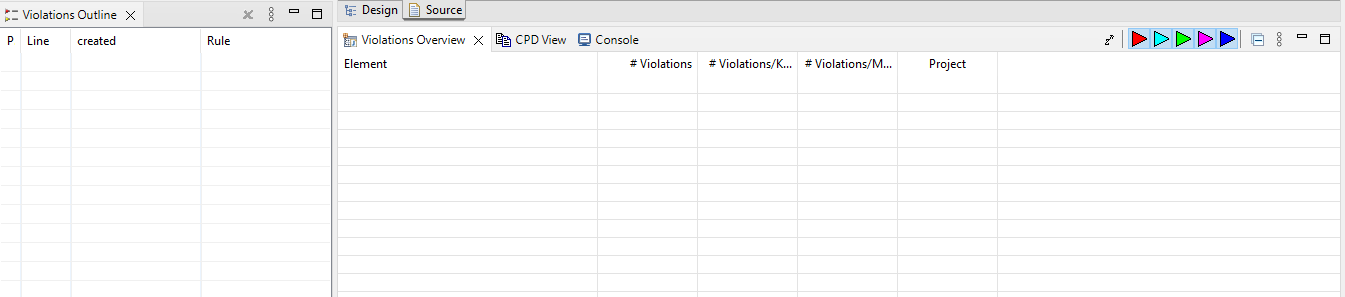


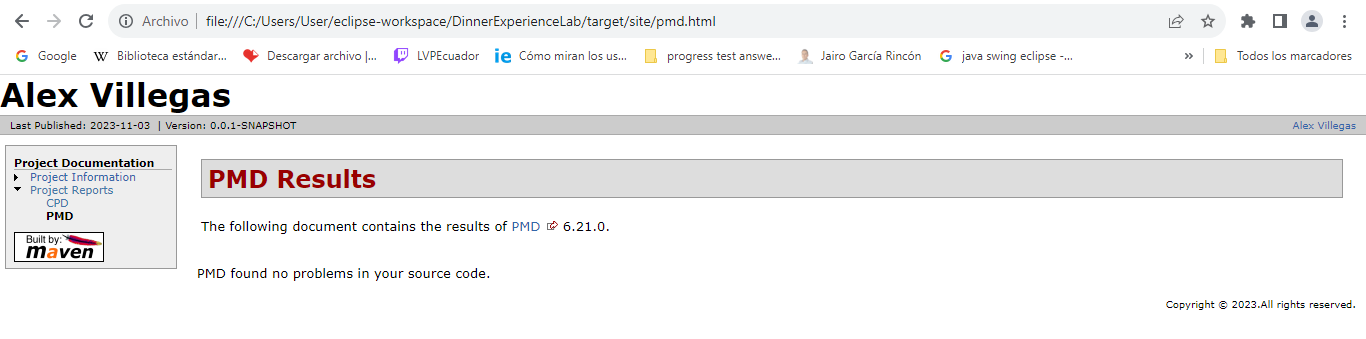






Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente



**package** dinnerExperience;

**import** java.util.Scanner;

@SuppressWarnings("PMD.UseUtilityClass")

**public** **class** DiningExperienceManager {

**private** **static** **final** **int** ***MAX\_ORDER\_QUANTITY*** = 100;

**public** **static** **void** main(String[] args) {

**try** ( Scanner scanner = **new** Scanner(System.***in***)) {

Meal[] menu = {

**new** Meal("Spaghetti", 8.0),

**new** Meal("Pizza", 10.0),

**new** Meal("Sweet and Sour Chicken", 9.0),

**new** Meal("Spring Rolls", 6.0),

**new** Meal("Tiramisu", 5.0),

**new** Meal("Cheesecake", 5.5),

**new** Meal("Chef's Special Soup", 7.5),

**new** Meal("Chef's Special Pasta", 12.0)

};

MenuManager menuManager = **new** MenuManager(menu);

menuManager.displayMenu();

System.***out***.println("Enter the quantity for each meal (0 to skip):");

**int**[] quantities = *getQuantities*(menu.length, scanner);

OrderManager orderManager = **new** OrderManager ();

Order order = orderManager.createOrder(quantities, menuManager.getMenu());

**double** totalCost = order. calculateTotalCost ();

OrderConfirmationManager confirmationManager = **new** OrderConfirmationManager ();

**if** (confirmationManager.confirmOrder(order, totalCost, scanner)) {

System.***out***.println("Order confirmed. Total Cost: $" + (**int**) totalCost);

} **else** {

System.***out***.println("Order canceled.");

}

} **catch** (Exception e) {

System.***out***.println("An error occurred: " + e.getMessage());

}

}

**private** **static** **int**[] getQuantities(**int** menuSize, Scanner scanner) {

**int**[] quantities = **new** **int**[menuSize];

**boolean** validInput = **false**;

**do** {

**try** {

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

System.***out***.print("Enter quantity for item " + (i + 1) + ": ");

quantities[i] = Integer.*parseInt*(scanner.nextLine());

**if** (quantities[i] < 1 || quantities[i] > DiningExperienceManager.***MAX\_ORDER\_QUANTITY***) {

**throw** **new** IllegalArgumentException("Invalid quantity. Please enter a number between 1 and "

+ DiningExperienceManager.***MAX\_ORDER\_QUANTITY*** + ".");

}

}

validInput = **true**;

} **catch** (NumberFormatException e) {

System.***out***.println("Error: Invalid input. Please enter a valid quantity.");

} **catch** (IllegalArgumentException e) {

System.***out***.println("Error: " + e.getMessage());

}

} **while** (!validInput);

**return** quantities;

}

}

**package** dinnerExperience;

**public** **class** Discount10OrMoreOrder **extends** Order {

**private** **static** **final** **double** ***DISCOUNT\_RATE*** = 0.2; // 20% discount for orders with 10 or more items

@Override

**public** **double** calculateTotalCost() {

**double** totalCost = *getBaseCost*(); // Obtiene el base\_cost utilizando el método protegido

**for** (Meal meal : getItems().keySet()) {

**int** quantity = getItems().get(meal);

totalCost += meal.getPrice() \* quantity;

}

**int** totalQuantity = getTotalQuantity();

**if** (totalQuantity >= 10) {

totalCost \*= (1 - ***DISCOUNT\_RATE***);

}

**return** totalCost;

}

}

**package** dinnerExperience;

**public** **class** Discount5OrMoreOrder **extends** Order {

**private** **static** **final** **double** ***DISCOUNT\_RATE*** = 0.1; // 10% discount for orders with 5 or more items

@Override

**public** **double** calculateTotalCost() {

**double** totalCost = *getBaseCost*(); // Obtiene el base\_cost utilizando el método protegido

**for** (Meal meal : getItems().keySet()) {

**int** quantity = getItems().get(meal);

totalCost += meal.getPrice() \* quantity;

}

**int** totalQuantity = getTotalQuantity();

**if** (totalQuantity >= 5) {

totalCost \*= (1 - ***DISCOUNT\_RATE***);

}

**return** totalCost;

}

}

**package** dinnerExperience;

**public** **class** Meal {

**private** String name;

**private** **double** price;

**public** Meal(String name, **double** price) {

**this**.name = name;

**this**.price = price;

}

**public** String getName() {

**return** name;

}

**public** **double** getPrice() {

**return** price;

}

}

/\*\*

\*

\*/

package dinnerExperience;

/\*\*

\*

\*/

public class MenuManager {

private Meal[] menu;

public MenuManager(Meal... menu) {

this.menu = menu.clone();

}

public void displayMenu() {

System.out.println("Menu Options:");

for (int i = 0; i < menu.length; i++) {

System.out.println((i + 1) + ". " + menu[i].getName() + " - $" + menu[i].getPrice());

}

}

public Meal[] getMenu() {

return menu == null ? null : menu.clone();

}

}

**package** dinnerExperience;

**public** **class** NormalOrder **extends** Order {

@Override

**public** **double** calculateTotalCost() {

**double** totalCost = *getBaseCost*(); // Obtiene el base\_cost utilizando el método protegido

**for** (Meal meal : getItems().keySet()) {

**int** quantity = getItems().get(meal);

totalCost += meal.getPrice() \* quantity;

}

**return** totalCost;

}

}

package dinnerExperience;

import java.util.Map;

import java.util.HashMap;

abstract class Order {

private static final double BASE\_COST = 5.0; // Base cost for every order

private Map<Meal, Integer> items = new HashMap<>();

protected void addItem(Meal meal, int quantity) {

items.put(meal, quantity);

}

protected Map<Meal, Integer> getItems() {

return items;

}

// Calculate the total cost for the order based on the chosen strategy

public abstract double calculateTotalCost();

// Calculate the total quantity of meals ordered

protected int getTotalQuantity() {

return getItems().values().stream().mapToInt(Integer::intValue).sum();

}

// Get the base cost for the order (accessible within the package and subclasses)

protected static double getBaseCost() {

return BASE\_COST;

}

}

/\*\*

\*

\*/

package dinnerExperience;

import java.util.Scanner;

/\*\*

\*

\*/

public class OrderConfirmationManager {

public boolean confirmOrder(Order order, double totalCost, Scanner scanner) {

while (true) {

try {

System.out.println("Order Summary:");

for (Meal meal : order.getItems().keySet()) {

int quantity = order.getItems().get(meal);

System.out.println(meal.getName() + " x " + quantity + " - $" + meal.getPrice() \* quantity);

}

System.out.println("Total Cost: $" + totalCost);

System.out.println("Do you want to confirm the order? (Y/N):");

String choice = scanner.next().toUpperCase();

if ("Y".equals(choice)) {

return true;

} else if ("N".equals(choice)) {

return false;

} else {

System.out.println("Invalid input. Please enter 'Y' to confirm or 'N' to cancel.");

}

} catch (Exception e) {

System.out.println("An error occurred while processing your order. Please try again.");

scanner.nextLine(); // Limpiar el buffer del scanner para evitar un bucle infinito

}

}

}

}

/\*\*

\*

\*/

package dinnerExperience;

/\*\*

\*

\*/

public class OrderManager {

private static final int MAX\_ORDER\_QUANTITY = 100;

public Order createOrder(int[] quantities, Meal... menu) {

Order order = new NormalOrder();

try {

if (menu == null || quantities == null) {

throw new NullPointerException("Menu and quantities must not be null.");

}

for (int i = 0; i < menu.length; i++) {

if (menu[i] == null) {

throw new IllegalArgumentException("Menu contains a null item at index: " + i);

}

if (quantities[i] > 0) {

if (quantities[i] > MAX\_ORDER\_QUANTITY) {

throw new IllegalArgumentException("Maximum order quantity for " + menu[i].getName() + " is " + MAX\_ORDER\_QUANTITY + ".");

}

order.addItem(menu[i], quantities[i]);

}

}

} catch (NullPointerException e) {

System.out.println("Error: " + e.getMessage());

// Puedes decidir cómo manejar la excepción aquí, como asignar un valor predeterminado a order o mostrar un mensaje de error al usuario.

} catch (IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

// Puedes decidir cómo manejar la excepción aquí, como asignar un valor predeterminado a order o mostrar un mensaje de error al usuario.

}

return order;

}

}

/\*\*

\*

\*/

package dinnerExperience;

/\*\*

\*

\*/

@SuppressWarnings("PMD.UseUtilityClass")

public class SpecialDiscounts {

private static final double SPECIAL\_OFFER\_DISCOUNT\_50 = 10.0;

private static final double SPECIAL\_OFFER\_DISCOUNT\_100 = 25.0;

public static double applySpecialOfferDiscount(double totalCost) {

double discountedCost = totalCost; // Create a separate variable to store the discounted value

if (totalCost > 100) {

discountedCost -= SPECIAL\_OFFER\_DISCOUNT\_100;

} else if (totalCost > 50) {

discountedCost -= SPECIAL\_OFFER\_DISCOUNT\_50;

}

return discountedCost; // Return the discounted value without modifying the original parameter

}

}

# Challenge

## Development

//Copyright (C) 2023

//All rights reserved

package maven3;

public class Vacation {

private String destination;

private int travelers;

private int duration;

private int base\_cost = 1000;

private double total\_cost = 0;

private int extra;

/\*\*

Constructor of the class \*

\*/

public Vacation() {

}

public String getDestination() {

return destination;

}

public void setDestination(String destination) {

this.destination = destination;

}

public int getTravelers() {

return travelers;

}

public void setTravelers(int travelers) {

this.travelers = travelers;

}

public int getDuration() {

return duration;

}

public void setDuration(int duration) {

this.duration = duration;

}

public int getBase\_cost() {

return base\_cost;

}

public void setBase\_cost(int base\_cost) {

this.base\_cost = base\_cost;

}

/\*\*

Return the total cost of the package \*

\*/

public double getTotal\_cost() {

if (total\_cost == 0){

return -1;

}

return total\_cost;

}

public void setTotal\_cost(double total\_cost) {

this.total\_cost = total\_cost;

}

public int getExtra() {

return extra;

}

public void setExtra(int extra) {

this.extra = extra;

}

/\*\*

Method responsible for verifying if the destination is a touristic

post in order to add an additional cost

@param destination is the name of the destiny of the package

\*\*/

public double verifyDestiny(String destination) {

int additional = 0;

if(destination.toUpperCase().equals("Paris")) {

additional = 500 + this.base\_cost;

}

else if(destination.toUpperCase().equals("New York City")) {

additional= 600 + this.base\_cost;

}

return additional + this.base\_cost;

}

/\*\* Method that return the total cost of the package including all the restrictions

@param totalcost is the total amount of money the package is going to cost

@param travelers is the number of the travelers that are going to use the package

@param duration is the time in days that the travel is going to last

\*\*/

public void getCost(double totalcost, int travelers, int duration) {

double discount = 0;

double fee = 200;

if(travelers>4 && travelers <10) {

discount = total\_cost\*0.10;

this.total\_cost = totalcost - discount;

}

if(travelers>10) {

discount = total\_cost\*0.20;

this.total\_cost = totalcost - discount;

}

if(duration<7) {

this.total\_cost = totalcost + fee;

}

if(duration>30 || travelers == 2) {

this.total\_cost = totalcost - fee;

}

else {

this.total\_cost = this.base\_cost;

}

}

/\*\* Method that calculate the total cost of the package including the extra cost

@param extraCode is the option or number that correspond to an specific add-on

@param travelers is the number of the travelers that are going to use the package

\*\*/

public void getTotalWithExtra(int extraCode, int travelers) {

double plusCost = 0;

if(extraCode<1 && extraCode > 3) {

System.out.println(-2);

}

switch(extraCode) {

case 1: plusCost = 200\*travelers; break;

case 2: plusCost = 150\*travelers; break;

case 3: plusCost = 100\*travelers; break;

default: plusCost = 0; break;

}

this.total\_cost = this.total\_cost+plusCost;

}

}

//Copyright (C) 2023

//All rights reserved

package maven3;

import java.util.Scanner;

public class VacationEstimator {

//CHECKSTYLE:OFF

public static void main(String[] args) {

//CHECKSTYLE:ON

try (Scanner sc = new Scanner(System.in)) {

Vacation vac1 = new Vacation();

System.out.println("\*\*\*\*\*Vacation Package Estimator\*\*\*\*\*");

// Manejar la entrada del usuario para el destino

System.out.println("Enter your destination:");

vac1.setDestination(getUserInput(sc));

// Manejar la entrada del usuario para el número de viajeros

System.out.println("Enter the number of travelers:");

vac1.setTravelers(getPositiveIntegerInput(sc));

// Manejar la entrada del usuario para la duración en días

System.out.println("Enter the duration in days:");

vac1.setDuration(getPositiveIntegerInput(sc));

// Manejar la entrada del usuario para los extras

printAddonOptions();

vac1.setExtra(getAddonChoice(sc));

while (vac1.getExtra() < 1 || vac1.getExtra() > 4) {

System.out.println("Invalid choice. Please select a valid option (1-4):");

printAddonOptions();

vac1.setExtra(getAddonChoice(sc));

}

// Validar el número de viajeros

if (vac1.getTravelers() > 80) {

System.out.println("The vacation package is not available for groups of more than 80 persons");

} else {

try {

double total\_cost = vac1.verifyDestiny(vac1.getDestination());

vac1.getCost(total\_cost, vac1.getTravelers(), vac1.getDuration());

vac1.getTotalWithExtra(vac1.getExtra(), vac1.getTravelers());

System.out.println("The total cost of the vacation package is " + vac1.getTotal\_cost());

} catch (Exception e) {

System.out.println("Error occurred while calculating the total cost: " + e.getMessage());

}

}

} catch (Exception e) {

System.out.println("An error occurred: " + e.getMessage());

}

}

private static String getUserInput(Scanner scanner) {

String userInput = null;

boolean validInput = false;

while (!validInput) {

userInput = scanner.nextLine();

if (!userInput.isEmpty()) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid string:");

}

}

return userInput;

}

private static int getPositiveIntegerInput(Scanner scanner) {

int input = 0;

boolean validInput = false;

while (!validInput) {

try {

input = Integer.parseInt(scanner.nextLine());

if (input > 0) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a positive number:");

}

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid number:");

}

}

return input;

}

private static void printAddonOptions() {

System.out.println("Select your add-ons:");

System.out.println("1. All-Inclusive Package - $200 per traveler");

System.out.println("2. Adventure Activities Package - $150 per traveler");

System.out.println("3. Spa and Wellness Package - $100 per traveler");

System.out.println("4. Nothing");

}

private static int getAddonChoice(Scanner scanner) {

int choice = 0;

boolean validInput = false;

while (!validInput) {

choice = getPositiveIntegerInput(scanner);

if (choice >= 1 && choice <= 4) {

validInput = true;

} else {

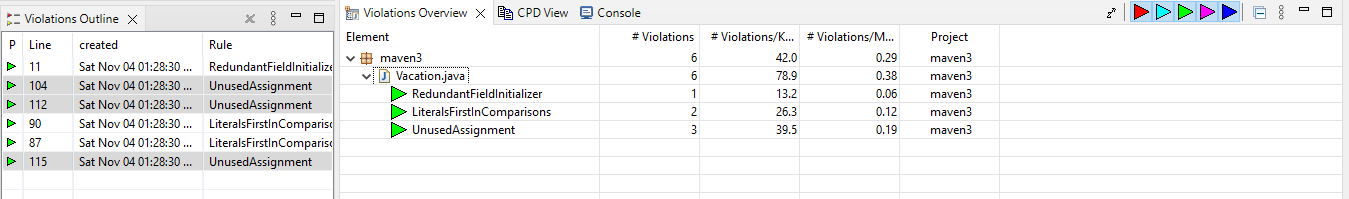
System.out.println("Invalid choice. Please select a valid option (1-4):");

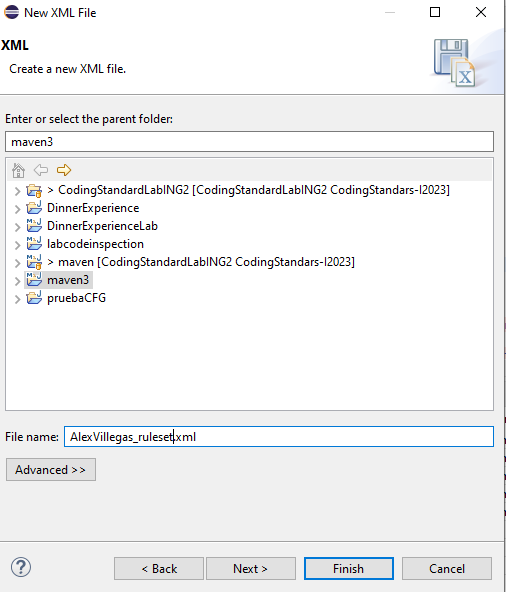
}

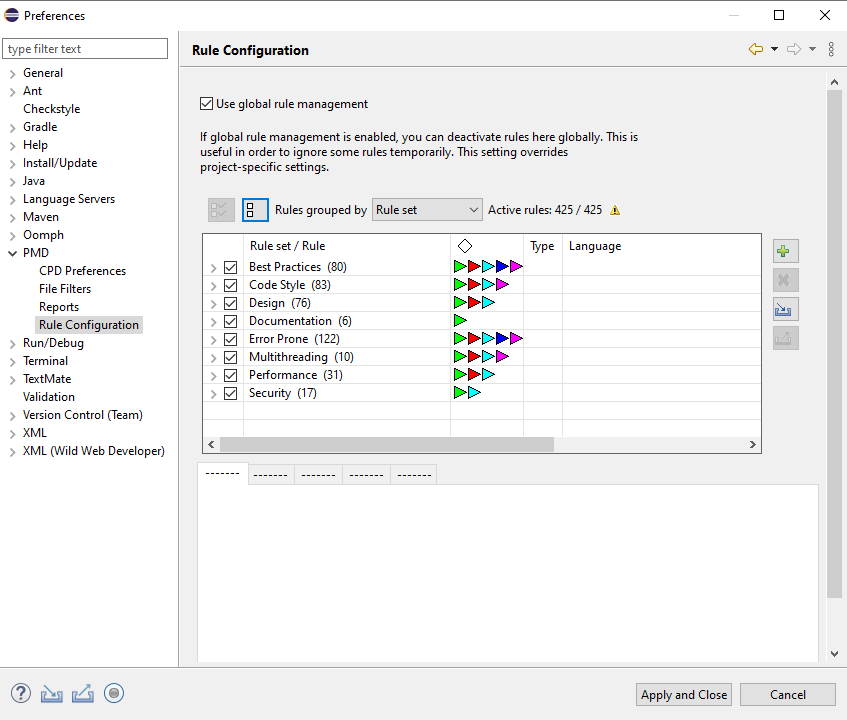
}

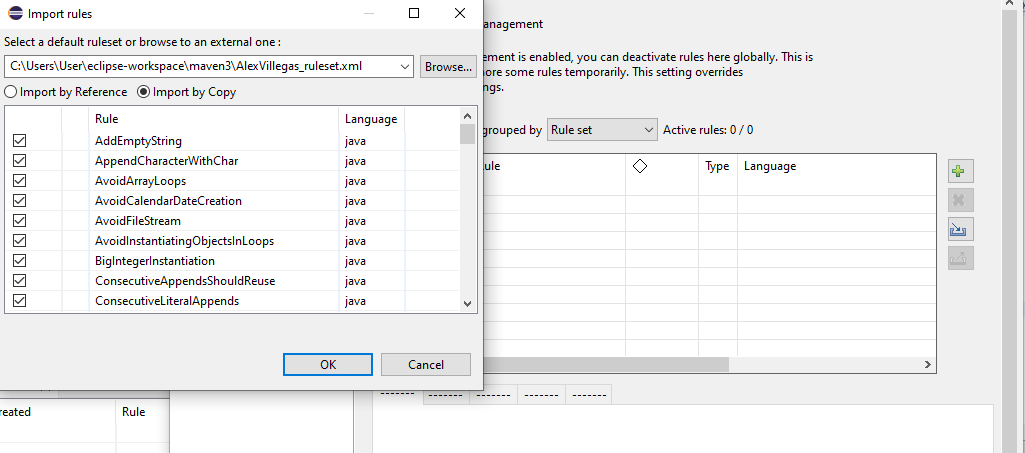
return choice;

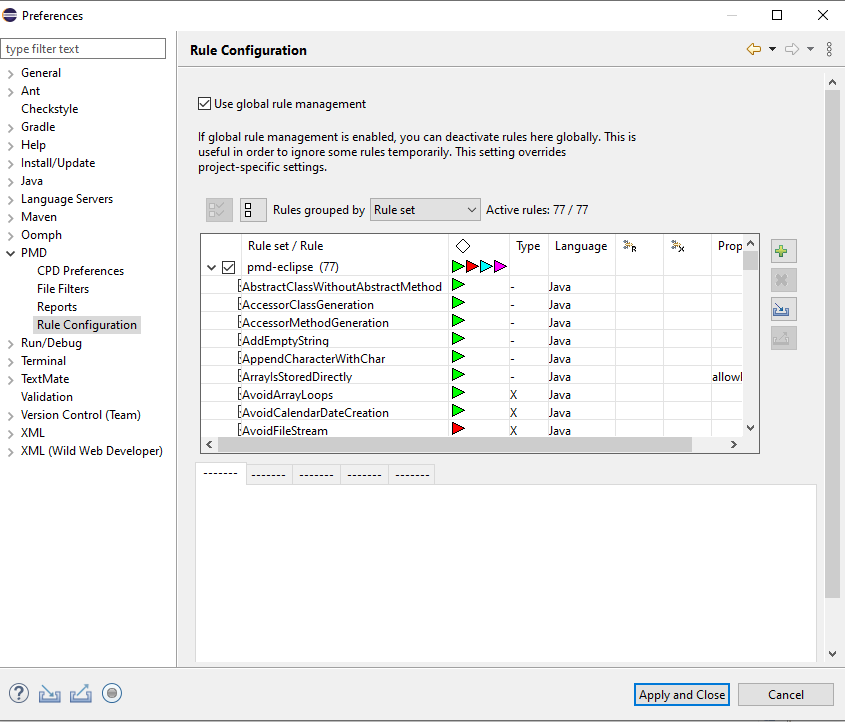
}

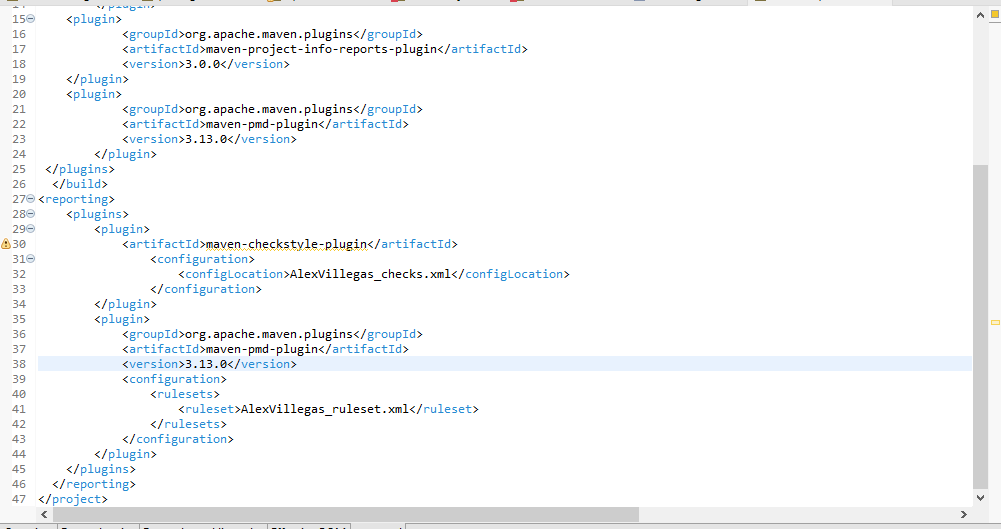
}

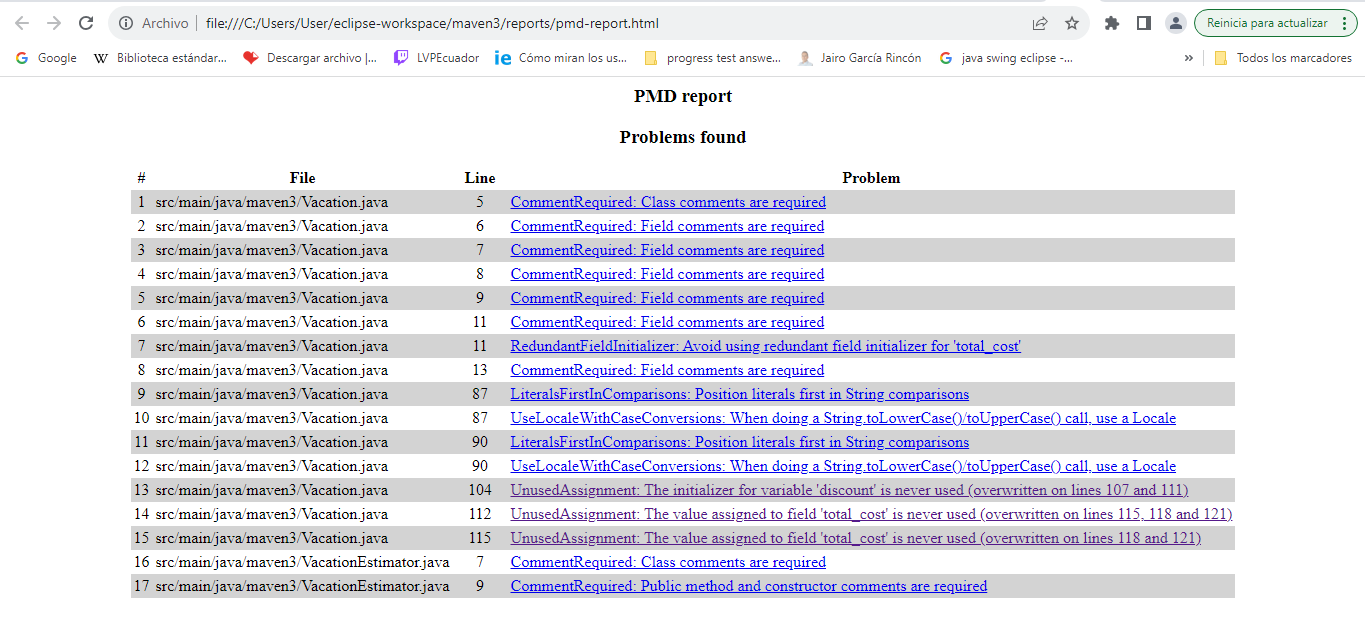


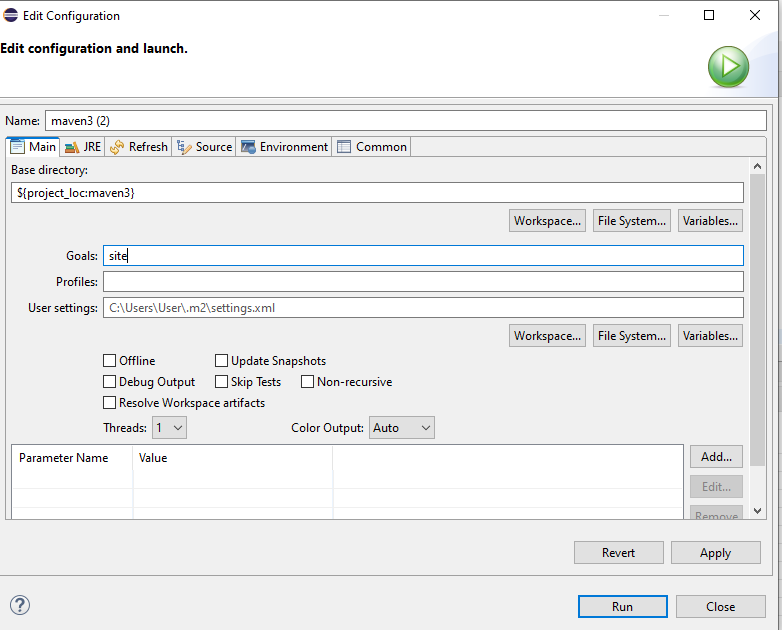








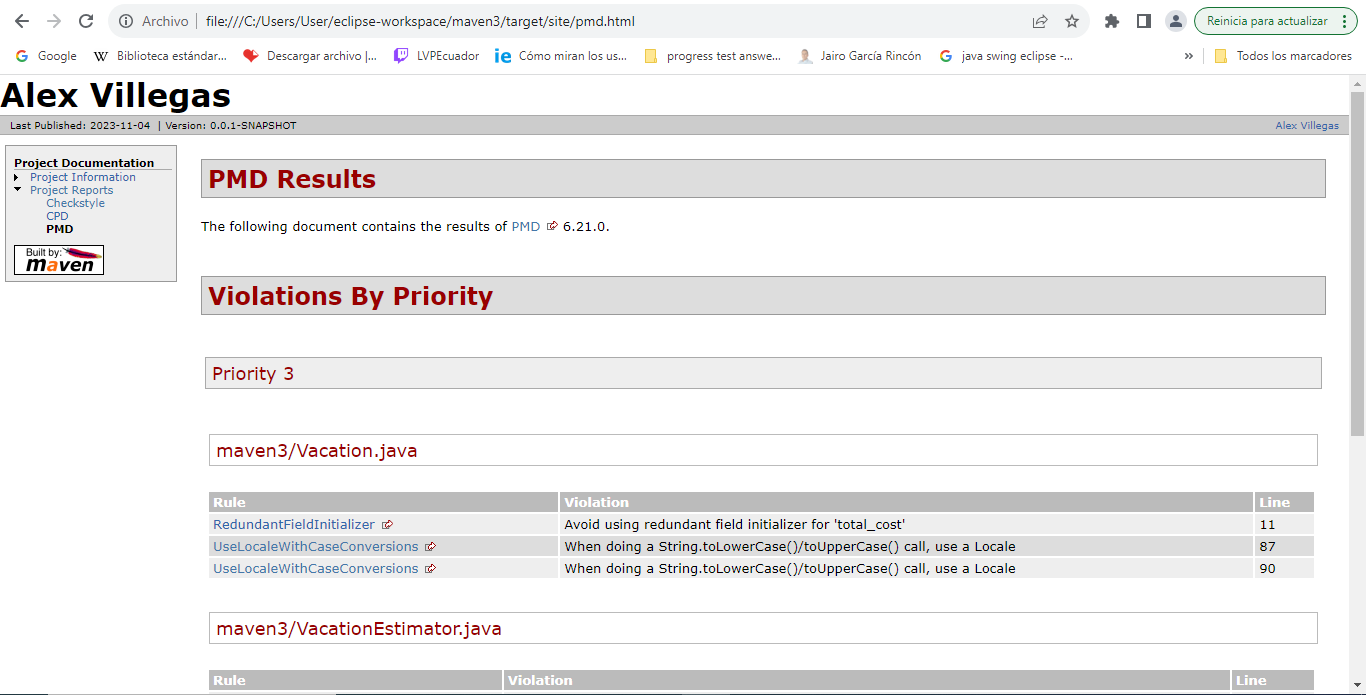




Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteTexto

Descripción generada automáticamente



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

//Copyright (C) 2023

//All rights reserved

package maven3;

import java.util.Locale;

/\*\*

\* This class represents a vacation package with information about the destination, the number of travelers, and the duration.

\* It also calculates the total cost of the vacation package based on various restrictions and selected add-ons.

\*

\* @author User

\* @version 1.0

\* @since 2023

\*/

public class Vacation {

//Class Fields

/\*\*

\* The destination of the vacation package.

\*/

private String destination;

/\*\*

\* The number of travelers for the vacation package.

\*/

private int travelers;

/\*\*

\* The duration in days of the vacation package.

\*/

private int duration;

/\*\*

\* The base cost of the vacation package.

\*/

private int base\_cost = 1000;

/\*\*

\* The total cost of the vacation package, including all additional charges and discounts.

\*/

private double total\_cost;

/\*\*

\* The code representing the selected extra/add-on for the vacation package.

\*/

private int extra;

/\*\*

Constructor of the class \*

\*/

public Vacation() {

}

public String getDestination() {

return destination;

}

/\*\*

\* Sets the destination of the vacation package.

\*

\* @param destination The name of the destination for the package.

\*/

public void setDestination(String destination) {

this.destination = destination;

}

public int getTravelers() {

return travelers;

}

/\*\*

\* Sets the number of travelers for the vacation package.

\*

\* @param travelers The number of travelers.

\*/

public void setTravelers(int travelers) {

this.travelers = travelers;

}

public int getDuration() {

return duration;

}

/\*\*

\* Sets the duration in days of the vacation package.

\*

\* @param duration The duration in days of the package.

\*/

public void setDuration(int duration) {

this.duration = duration;

}

public int getBase\_cost() {

return base\_cost;

}

/\*\*

\* Sets the base cost of the vacation package.

\*

\* @param base\_cost The base cost of the package.

\*/

public void setBase\_cost(int base\_cost) {

this.base\_cost = base\_cost;

}

/\*\*

Return the total cost of the package \*

\*/

public double getTotal\_cost() {

if (total\_cost == 0){

return -1;

}

return total\_cost;

}

public void setTotal\_cost(double total\_cost) {

this.total\_cost = total\_cost;

}

public int getExtra() {

return extra;

}

/\*\*

\* Sets the code of the selected add-on for the vacation package.

\*

\* @param extra The code of the selected add-on.

\*/

public void setExtra(int extra) {

this.extra = extra;

}

/\*\*

Method responsible for verifying if the destination is a touristic

post in order to add an additional cost

@param destination is the name of the destiny of the package

\*\*/

public double verifyDestiny(String destination) {

int additional = 0;

if("Paris".equals(destination.toUpperCase(Locale.ROOT))) {

additional = 500 + this.base\_cost;

}

else if("New York City".equals(destination.toUpperCase(Locale.ROOT))) {

additional= 600 + this.base\_cost;

}

return additional + this.base\_cost;

}

/\*\* Method that return the total cost of the package including all the restrictions

@param totalcost is the total amount of money the package is going to cost

@param travelers is the number of the travelers that are going to use the package

@param duration is the time in days that the travel is going to last

\*\*/

public void getCost(double totalcost, int travelers, int duration) {

double discount;

double fee = 200;

if(travelers>4 && travelers <10) {

discount = total\_cost\*0.10;

this.total\_cost = totalcost - discount;

}

if(travelers>10) {

discount = total\_cost\*0.20;

this.total\_cost = totalcost - discount; //NOPMD

}

if(duration<7) {

this.total\_cost = totalcost + fee; //NOPMD

}

if(duration>30 || travelers == 2) {

this.total\_cost = totalcost - fee; //NOPMD

}

else {

this.total\_cost = this.base\_cost;

}

}

/\*\* Method that calculate the total cost of the package including the extra cost

@param extraCode is the option or number that correspond to an specific add-on

@param travelers is the number of the travelers that are going to use the package

\*\*/

public void getTotalWithExtra(int extraCode, int travelers) {

double plusCost = 0;

if(extraCode<1 && extraCode > 3) {

System.out.println(-2);

}

switch(extraCode) {

case 1: plusCost = 200\*travelers; break;

case 2: plusCost = 150\*travelers; break;

case 3: plusCost = 100\*travelers; break;

default: plusCost = 0; break;

}

this.total\_cost = this.total\_cost+plusCost;

}

}

//Copyright (C) 2023

//All rights reserved

package maven3;

import java.util.Scanner;

/\*\*

\* VacationEstimator class estimates the cost of a vacation package based on user input for destination, number of travelers,

\* duration, and selected add-ons. It handles user input, validates inputs, and calculates the total cost of the vacation package.

\*

\* @version 1.0

\* @since 2023

\*/

public class VacationEstimator {

//CHECKSTYLE:OFF

/\*\*

\* Main method initializes the vacation package estimator. It prompts the user for input, validates the input,

\* calculates the total cost, and displays the result.

\*

\* @param args The command-line arguments (not used).

\*/

public static void main(String[] args) {

//CHECKSTYLE:ON

try (Scanner sc = new Scanner(System.in)) {

Vacation vac1 = new Vacation();

System.out.println("\*\*\*\*\*Vacation Package Estimator\*\*\*\*\*");

// Manejar la entrada del usuario para el destino

System.out.println("Enter your destination:");

vac1.setDestination(getUserInput(sc));

// Manejar la entrada del usuario para el número de viajeros

System.out.println("Enter the number of travelers:");

vac1.setTravelers(getPositiveIntegerInput(sc));

// Manejar la entrada del usuario para la duración en días

System.out.println("Enter the duration in days:");

vac1.setDuration(getPositiveIntegerInput(sc));

// Manejar la entrada del usuario para los extras

printAddonOptions();

vac1.setExtra(getAddonChoice(sc));

while (vac1.getExtra() < 1 || vac1.getExtra() > 4) {

System.out.println("Invalid choice. Please select a valid option (1-4):");

printAddonOptions();

vac1.setExtra(getAddonChoice(sc));

}

// Validar el número de viajeros

if (vac1.getTravelers() > 80) {

System.out.println("The vacation package is not available for groups of more than 80 persons");

} else {

try {

double total\_cost = vac1.verifyDestiny(vac1.getDestination());

vac1.getCost(total\_cost, vac1.getTravelers(), vac1.getDuration());

vac1.getTotalWithExtra(vac1.getExtra(), vac1.getTravelers());

System.out.println("The total cost of the vacation package is " + vac1.getTotal\_cost());

} catch (Exception e) {

System.out.println("Error occurred while calculating the total cost: " + e.getMessage());

}

}

} catch (Exception e) {

System.out.println("An error occurred: " + e.getMessage());

}

}

private static String getUserInput(Scanner scanner) {

String userInput = null;

boolean validInput = false;

while (!validInput) {

userInput = scanner.nextLine();

if (!userInput.isEmpty()) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a valid string:");

}

}

return userInput;

}

private static int getPositiveIntegerInput(Scanner scanner) {

int input = 0;

boolean validInput = false;

while (!validInput) {

try {

input = Integer.parseInt(scanner.nextLine());

if (input > 0) {

validInput = true;

} else {

System.out.println("Invalid input. Please enter a positive number:");

}

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid number:");

}

}

return input;

}

private static void printAddonOptions() {

System.out.println("Select your add-ons:");

System.out.println("1. All-Inclusive Package - $200 per traveler");

System.out.println("2. Adventure Activities Package - $150 per traveler");

System.out.println("3. Spa and Wellness Package - $100 per traveler");

System.out.println("4. Nothing");

}

private static int getAddonChoice(Scanner scanner) {

int choice = 0;

boolean validInput = false;

while (!validInput) {

choice = getPositiveIntegerInput(scanner);

if (choice >= 1 && choice <= 4) {

validInput = true;

} else {

System.out.println("Invalid choice. Please select a valid option (1-4):");

}

}

return choice;

}

}

# Conclusion

In conclusion, our exploration of code inspection and PMD has equipped us with valuable skills and tools to create high-quality, robust, and maintainable software. By integrating these practices into our development workflows, we can contribute to the creation of reliable software solutions that meet the needs of end-users while being efficient and sustainable in the long run.

# Recommendations

1. **Regular PMD Integration:** Integrate PMD into the continuous integration (CI) pipeline, enabling automated code analysis with every build. Regular analysis ensures ongoing code quality maintenance.
2. **Custom Rule Configuration:** Encourage developers to configure custom PMD rules tailored to the project's specific requirements. Custom rules enhance the tool's effectiveness in catching project-specific issues.
3. **Educational Workshops:** Conduct workshops and training sessions focusing on PMD's advanced features, helping developers harness its full potential for in-depth code analysis.
4. **Code Inspection Guidelines:** Establish clear code inspection guidelines within the development team. Define specific criteria and severity levels for issues detected by PMD to streamline the resolution process.
5. **Documentation and Knowledge Sharing:** Maintain comprehensive documentation about common coding issues detected by PMD. Foster knowledge sharing sessions where team members can discuss and learn from resolved issues.
6. **Regular Tool Updates:** Ensure that PMD is regularly updated to the latest version to benefit from bug fixes, new features, and improved rule sets. Stay abreast of PMD's advancements in the field of static code analysis.
7. **Code Review Best Practices:** Reinforce manual code review best practices alongside PMD usage. Emphasize the value of human expertise in identifying nuanced issues that automated tools might miss.

# References

[1] “What is code inspection in software engineering ?,” GeeksforGeeks, https://www.geeksforgeeks.org/what-is-code-inspection-in-software-engineering/ (accessed Nov. 5, 2023).

[2] Pmd, PMD, https://pmd.github.io/ (accessed Nov. 5, 2023).

# Resources

***Link of the public repository:*** [***https://github.com/ajvilleg10/WorkshopCodingInspectionING2***](https://github.com/ajvilleg10/WorkshopCodingInspectionING2)

***Link of the public repository (challenge):*** [***https://github.com/ajvilleg10/WorkshopCodingStandar***](https://github.com/ajvilleg10/WorkshopCodingStandar)