

Single Responsibility Principle

INTRO

Ogni singola entità sw come ad esempio una classe oppure un metodo dovrebbe avere una e una solo ragione per cambiare.

Se ci sono 2 o più ragioni per cambiare una data classe/metodo, allora questo è un segno che non stiamo rispettando il principio di "Singola Responsabilità".

Quindi se una classe/metodo esegue più operazioni o procedure è fortemente consigliato separare questi in più classi/metodi.

Segue che ogni classe/metodo dovrebbe preoccuparsi/gestire una singola responsabilità.

Questo può essere fatto con l'aiuto dell'astrazione: ovvero tramite interfacce o classi.

Questo ci aiuterà ad ottenere componenti sw indipendenti/disaccoppiati tra loro.

ACCOPPIAMENTO SW

Esistono 2 tipi di accoppiamento (coupling) tra entità sw:

(Nota per entità sw: intendiamo classi/metodi)

:(- **Accoppiamento stretto** (Tight Coupling)

- i) Entità sw dipendono fortemente una dall'altra.
Questo significa che poichè le entità sw sono strettamente dipendenti cambiandone una, questi cambiamenti si ripercuotono anche sull'altra entità sw.
- ii) Succede tipicamente le entità sw hanno molte responsabilità

:) - **Accoppiamento basso** (Loosely Coupling)

- i) Entità sw sono indipendenti l'una dall'altra.
Questo vuol dire ad es che se cambiamo il comportamento di una classe questa non andrà ad impattare sull'altra perchè sono appunto indipendenti.
- ii) Interfacce (astrazioni) sono strumenti molto potenti per disaccoppiare entità sw.

Quindi la parola d'ordine è: **INTERFACCE**

- iii) Accoppiamento basso può essere ottenuto a pieno applicando i principi SOLID.

Consideriamo il seguente codice:

```
public class App {  
  
    public static void main(String[] args) {  
        System.out.println("Welcome to App");  
  
        // 1. Acquisizione dati utente da tastiera  
        Scanner scanner = new Scanner(System.in);  
  
        // 1a. Acquisizione primo dato utente  
        System.out.println("Enter first number");  
        String input1 = scanner.nextLine();  
  
        // 1b. Acquisizione secondo dato utente  
        System.out.println("Enter first number");  
        String input2 = scanner.nextLine();  
  
        int numb1;  
        int numb2;  
  
        // 2. Check validazione input utente  
        // 2a. Check validazione primo input utente  
        if(input1 == null) {  
            System.out.println("First number is not valid...");  
            return;  
        }  
        try {  
            numb1 = Integer.parseInt(input1);  
        } catch(NumberFormatException ex) {  
            System.out.println("First number is not valid...");  
            return;  
        }  
  
        // 2b. Check validazione secondo input utente  
        if(input2 == null) {  
            System.out.println("Second number is not valid...");  
            return;  
        }  
        try {  
            numb2 = Integer.parseInt(input2);  
        } catch(NumberFormatException ex) {  
            System.out.println("Second number is not valid...");  
            return;  
        }  
  
        // 3. Elaborazione matematica  
        int result = numb1 + nyumb2;  
  
        // 4. Comunicazione del risultato  
        System.out.println("The result is: " + result);  
        System.out.println("End of App");  
    }  
}
```

Il codice mostrato appena sopra è evidente che sta violando il principio di

" SINGOLA RESPONSABILITA' " (SRP -> Single Responsibility Principle)

perchè nell'unico metodo (main) dell'unica classe (App) è stato incapsulato/affogato tutto il codice che è responsabile di almeno 4 funzionalità diverse e che sono:

- **Acquisizione** dati di input
- **Validazione** dati di input
- **Elaborazione** principale del programma
- **Comunicazione risultato**

Ci sono quindi multiple responsabilità all'interno di un dato metodo, il che significa che dovremo separare/splittare il comportamento di questo.

STEP 1 - RESPONSABILITA' ACQUISIZIONE INPUT

Possiamo creare una classe ad esempio: InputProcessor dove abbiamo un metodo che ha la responsabilità di ACQUISIRE I 2 NUMERI INPUT dall'utente:

```
public class InputProcessor {  
    public List<String> process() {  
        // 1. Acquisizione dati utente da tastiera  
        Scanner scanner = new Scanner(System.in);  
  
        // 1a. Acquisizione primo dato utente  
        System.out.println("Enter first number");  
        String input1 = scanner.nextLine();  
  
        // 1b. Acquisizione secondo dato utente  
        System.out.println("Enter first number");  
        String input2 = scanner.nextLine();  
  
        scanner.close();  
        return List.of( input1, input2 );  
    }  
}
```

Notiamo che questo metodo ha solo la responsabilità di acquisire i dati in input dall'utente.

Al momento non ci preoccupiamo della validità di questi dati inseriti dall'utente in quanto vi sarà un'altra entità sw che avrà la sola responsabilità di validare l'input acquisito da questa entità sw sopra rappresentata.

Vediamo in parallelo già come può essere modificato la nostra applicazione principale:

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("Welcome to App");  
  
        InputProcessor inputProcessor = new InputProcessor();  
        List<String> inputs = inputProcessor.process();  
  
        int numb1;  
        int numb2;  
  
        // 2. Check validazione input utente  
        // 2a. Check validazione primo input utente  
        if(inputs.get(0) == null) {  
            System.out.println("First number is not valid...");  
            return;  
        }  
        try {  
            numb1 = Integer.parseInt(inputs.get(0));  
        } catch(NumberFormatException ex) {  
            System.out.println("First number is not valid...");  
            return;  
        }  
  
        // 2b. Check validazione secondo input utente  
        if(inputs.get(1) == null) {  
            System.out.println("Second number is not valid...");  
            return;  
        }  
        try {  
            numb2 = Integer.parseInt(inputs.get(1));  
        } catch(NumberFormatException ex) {  
            System.out.println("Second number is not valid...");  
            return;  
        }  
  
        // 3. Elaborazione matematica  
        int result = numb1 + nyumb2;  
  
        // 4. Comunicazione del risultato  
        System.out.println("The result is: " + result);  
        System.out.println("End of App");  
    }  
}
```

STEP 2 - RESPONSABILITA' VALIDAZIONE INPUT

Possiamo creare una classe ad esempio: ViolationChecker dove abbiamo un metodo che ha la responsabilità di VALIDARE L'INPUT inserito dall'utente:

```
public class ViolationChecker {  
    public boolean isValid(List<String> inputs){  
        // 2. Check validazione input utente  
        if(inputs == null || inputs.size() < 2) {  
            return false;  
        }  
        // 2a. Check validazione primo input utente  
        if(inputs.get(0) == null) {  
            return false;  
        }  
        try {  
            Integer.parseInt(inputs.get(0));  
        } catch(NumberFormatException ex) {  
            return false;  
        }  
        // 2b. Check validazione secondo input utente  
        if(inputs.get(1) == null) {  
            return false;  
        }  
        try {  
            Integer.parseInt(inputs.get(1));  
        } catch(NumberFormatException ex) {  
            return false;  
        }  
        return true;  
    }  
}
```

Vediamo in parallelo già come può essere modificato la nostra applicazione principale:

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("Welcome to App");  
  
        InputProcessor inputProcessor = new InputProcessor();  
        List<String> inputs = inputProcessor.process();  
  
        ViolationChecker violationChecker = new ViolationChecker();  
        boolean isInputsValid = violationChecker.isValid(inputs);  
  
        if(!isInputsValid) {  
            System.out.println("One of inputs is invalid...");  
            return;  
        }  
  
        int numb1 = Integer.parseInt(inputs.get(0));  
        int numb2 = Integer.parseInt(inputs.get(1));  
  
        // 3. Elaborazione matematica  
        int result = numb1 + nyumb2;  
  
        // 4. Comunicazione del risultato  
        System.out.println("The result is: " + result);  
        System.out.println("End of App");  
    }  
}
```


STEP 3 - RESPONSABILITA' ELABORAZIONE PRICIPALE

Possiamo creare una classe ad esempio: Operation dove abbiamo un metodo che ha la responsabilità di ELABORARE I DATI inseriti dall'utente:

```
public class Operation {  
    public int execute(int number1, int number2) {  
        return number1 + number2;  
    }  
}
```

Vediamo in parallelo già come può essere modificato la nostra applicazione principale:

```
public class App {  
  
    public static void main(String[] args) {  
        System.out.println("Welcome to App");  
  
        InputProcessor inputProcessor = new InputProcessor();  
        List<String> inputs = inputProcessor.process();  
  
        ViolationChecker violationChecker = new ViolationChecker();  
        boolean isInputsValid = violationChecker.isValid(inputs);  
  
        if(!isInputsValid) {  
            System.out.println("One of inputs is invalid...");  
            return;  
        }  
  
        int numb1 = Integer.parseInt(inputs.get(0));  
        int numb2 = Integer.parseInt(inputs.get(1));  
  
        Operation operation = new Operation();  
        int result = operation.execute(numb1, numb2);  
  
        // 4. Comunicazione del risultato  
        System.out.println("The result is: " + result);  
        System.out.println("End of App");  
    }  
}
```

STEP 4 - RESPONSABILITA' COMUNICAZIONE RISULTATO

Possiamo creare una classe ad esempio: PrinterResult dove abbiamo un metodo che ha la responsabilità di COMUNICARE A SCHERMO IL RISULTATO all'utente:

```
public class PrinterResult {  
    public void printScreen(int result) {  
        System.out.println("The result is:" + result);  
    }  
}
```

Vediamo in parallelo già come può essere modificato la nostra applicazione principale:

```
public class App {  
  
    public static void main(String[] args) {  
        System.out.println("Welcome to App");  
  
        InputProcessor inputProcessor = new InputProcessor();  
        List<String> inputs = inputProcessor.process();  
  
        ViolationChecker violationChecker = new ViolationChecker();  
        boolean isInputsValid = violationChecker.isValid(inputs);  
  
        if(!isInputsValid) {  
            System.out.println("One of inputs is invalid...");  
            return;  
        }  
  
        int num1 = Integer.parseInt(inputs.get(0));  
        int num2 = Integer.parseInt(inputs.get(1));  
  
        Operation operation = new Operation();  
        int result = operation.execute(num1, num2);  
  
        PrinterResult printerResult = new PrinterResult();  
        printerResult.printScreenResult(result);  
  
        System.out.println("End of App");  
    }  
}
```

Adesso ogni singola classe/metodo è responsabile di una sola funzionalità e quindi SRP è soddisfatto.

Nota: ogni metodo ha una sola ragione per cambiare in quanto incapsula solo una funzionalità di cui è responsabile.