****

**Scuola di**

**Ingegneria**

*Corso di Laurea Triennale in Ingegneria Informatica*

**Elaborato per Esame Orale Ingegneria del Software**

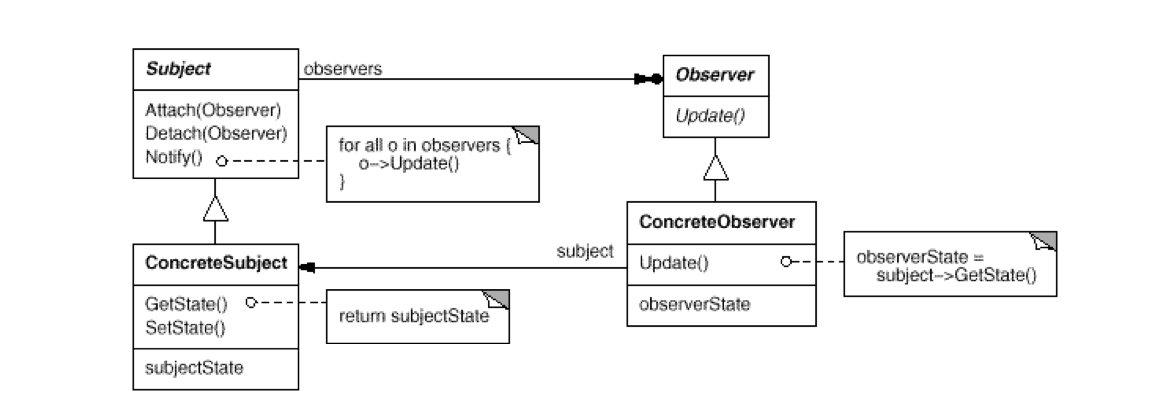
**10 luglio 2019**

**Pattern Mix)** Observer, Abstract Factory e Singleton.

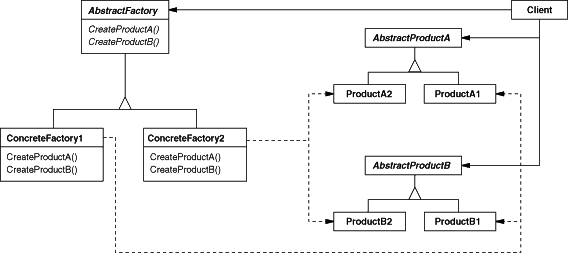
Edoardo Re

**596294**

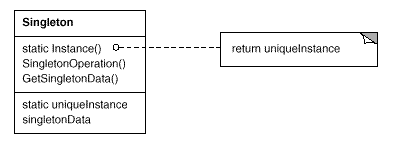
**Class Diagram di Observer:**

****

**Class Diagram di Abstract Factory:**

****

**Class Diagram di Singleton:**

****

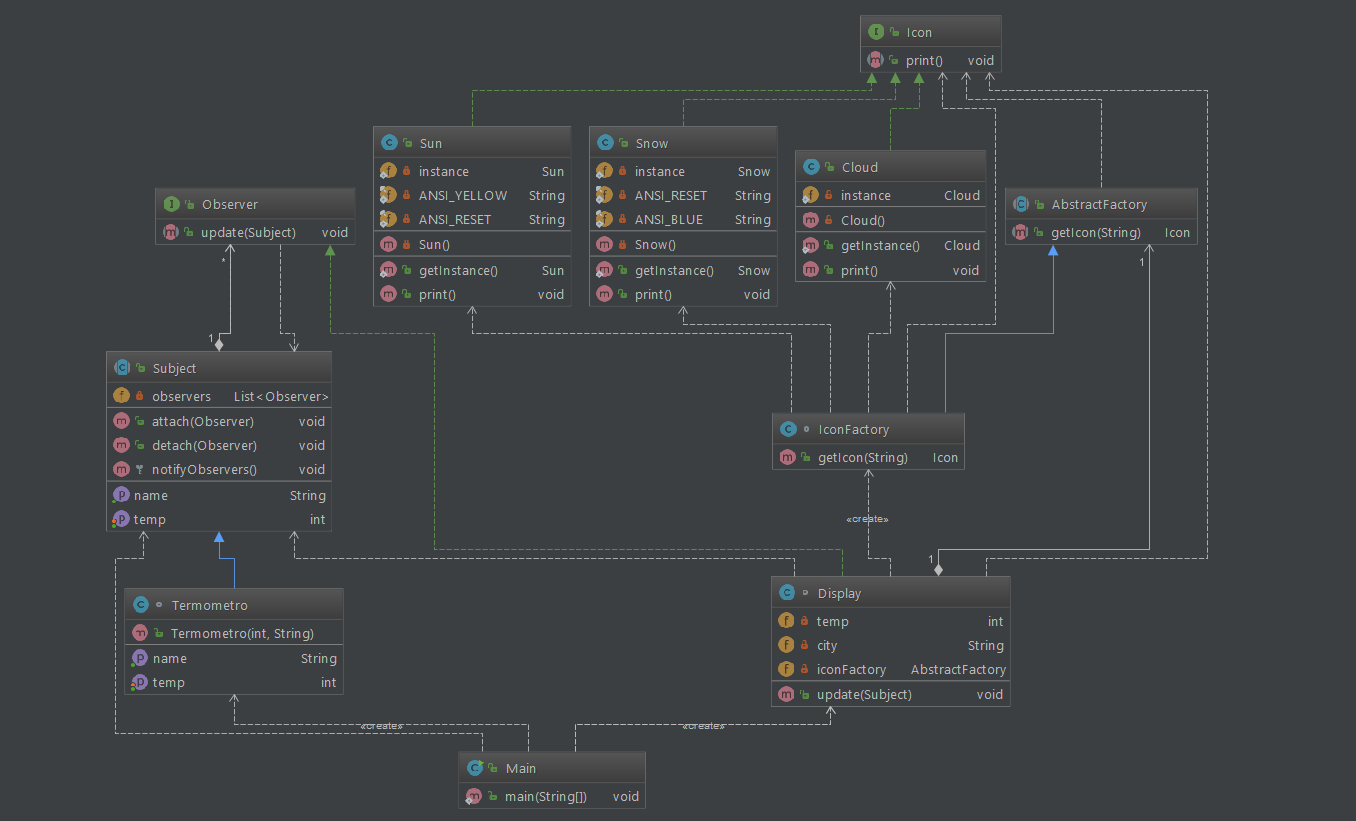
**Obiettivo:**

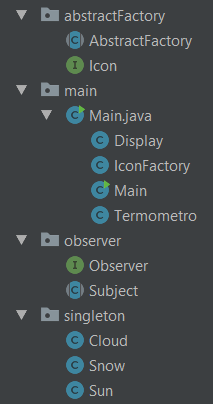
L’obiettivo dell’elaborato è quello far diventare più dinamico il pattern Abstract Factory grazie ad Observer; si vuole realizzare la creazione di prodotti che verranno istanziati una sola volta grazie al pattern Singleton. In questo caso è compito dell’Observer scegliere quale Product fare creare.

**Descrizione dell’esempio:**

L’elaborato unisce i tre pattern con un semplice esempio: Si vuole osservare la temperatura di più termometri sparsi in varie città. L’aggiornamento della temperatura dei termometri avviene “ogni ora” e viene osservata da un singolo Display che mostra il valore rilevato. Il display si avvale della collaborazione dell’Abstract Factory per creare un’icona, generata sulla base della temperatura rilevata. L’icona viene istanziata soltanto una volta grazie al pattern Singleton. I valori delle temperature nelle varie città sono generati in maniera casuale (in un range 0-40 gradi C) e vengono aggiornati “ogni ora” con un discostamento di + o – 5 gradi C. Il client si preoccupa solamente dei Termometri e del Display, disinteressandosi del resto.

**Class Diagram:**





*Le classi sono inserite all’interno di 4 package differenti che permettono di identificare meglio i 3 pattern utilizzati. La classe Termometro opera come Subject osservato dalla classe Display che a sua volta utilizza IconFactory per ricevere un’istanza di una Icon tra: Cloud, Snow, Sun. Si è scelto di realizzare Observer e Icon come interfacce come in molti casi di applicazione di questi due pattern. Il Singleton invece è visibile nelle classi Cloud, Snow e Sun che implementano l’interfaccia Icon.*

**Classi Partecipanti:**

AbstractFactory: definita appunto come classe astratta, mostra il metodo astratto *getIcon(String icon)* che deve essere implementato da chi estende tale classe.

Icon: definita come interfaccia espone il metodo *print().*

Display: implementa l’interfaccia Observer con il metodo *update(Stubject s)* che viene chiamato quando varia il valore misurato da uno dei termometri. In questo metodo si delega l’AbstractFactory per la creazione delle icone (sole, neve, nuvola) che vengono scelte in base al valore della temperatura osservata. In questo metodo si esegue la print dei valori e dell’icona appena viene istanziata.

IconFactory: estende la classe AbstractFactory implementando il metodo *getIcon(String iconType)*. In tale metodo si confronta la stringa passata nei parametri con delle stringhe preimpostate utili per comprendere quale ‘Prodotto’ (icona) istanziare.

Termometro: estende la classe Subject ed ha come attributi il nome della città e un intero che rappresenta la temperatura. Vengono definiti i metodi get dei due attributi per permettere a Display di poter richiedere i valori. Si è schelto di implementare il set soltanto per la temperatura, al suo interno si ha la chiamata di *notifyObservers()* che è definito nella superclass e invoca *update(this)* per tutti gli observers iscritti alla lista.

Observer: è una semplice interfaccia che espone il metodo *update(Subject s)* che viene implementato da Display.

Subject: è una classe astratta che tiene la lista degli Observer iscritti per essere notificati al cambiamento di temperatura. La lista è implementata grazie a *java.util.ArrayList* e *java.util.List*. Come di norma espone i metodi per iscriversi e disiscriversi a tale lista.

Cloud, Snow, Sun: sono tre classi che implementano Icon ed hanno il costruttore privato, dunque non si può istanziare alcun oggetto se non con il metodo *getInstance()* che però ne istanzia solo una all`inizio e ritorna successivamente sempre il riferimento a tale oggetto. (Somiglianza di AbstractFactory + Singleton al Proxy)

Main: definisce un array con delle città e crea per ognuna di esse un termometro (gestito semplicemente con un array di Termometro). Crea un Display che viene iscritto ad ogni lista di osservatori dei termometri; si simula il cambiamento di temperatura ogni 5 secondi con una variazione di quella precedente nell’intervallo di (-5, +5).

**Codice:**

Main.java file:

1. **package** main;
3. **import** abstractFactory.AbstractFactory;
4. **import** abstractFactory.Icon;
5. **import** observer.Observer;
6. **import** observer.Subject;
7. **import** singleton.Cloud;
8. **import** singleton.Snow;
9. **import** singleton.Sun;
11. **public** **class** Main {
12. **public** **static** **void** main(String[] args) **throws** InterruptedException {
13. String[] paesi = {"NewYork", "SanMarino", "Firenze", "Prato", "Pistoia", "Empoli", "Arezzo", "Viareggio",
14. "Sutri", "Siracusa", "Trento", "Rimini", "Genova", "Alassio", "Assisi", "Londra", "Sassari",
15. "Pontassieve", "Milano", "Roma", "Napoli", "Pisa", "Lucca"};
16. Subject[] termometri = **new** Termometro[paesi.length];
17. **for** (**int** i = 0; i < termometri.length; i++) {
18. termometri[i] = **new** Termometro((**int**) (Math.random() \* 40), paesi[i]);
19. }
20. Display display = **new** Display();
21. **for** (Subject t : termometri)
22. t.attach(display);
23. **int** h = 9;
24. **for** (**int** i = 0; i < 10; i++) {
25. System.out.println();
26. System.out.println("\u001B[31m" + "Rilevazione Meteo delle ore " + (h++) + ":00" + "\u001B[0m");
27. System.out.println();
28. **for** (Subject t : termometri) {
29. **if** ((**int**) (Math.random() \* 2) == 1)
30. t.setTemp(t.getTemp() - (**int**) (Math.random() \* 5));
31. **else**
32. t.setTemp(t.getTemp() + (**int**) (Math.random() \* 5));
33. }
34. Thread.sleep(5000);
35. }
36. **for** (Subject t : termometri)
37. t.detach(display);
38. }
39. }
41. **class** Termometro **extends** Subject {
42. **private** **int** temperatura;
43. **private** **final** String cityName;
45. **public** Termometro(**int** t, String n) {
46. temperatura = t;
47. cityName = n;
48. }
50. @Override
51. **public** **void** setTemp(**int** s) {
52. temperatura = s;
53. notifyObservers();
54. }
56. @Override
57. **public** **int** getTemp() {
58. **return** temperatura;
59. }

62. @Override
63. **public** String getName() {
64. **return** cityName;
65. }
66. }
68. **class** Display **implements** Observer {
69. **private** **int** temp;
70. **private** String city;
71. **private** AbstractFactory iconFactory;
73. @Override
74. **public** **void** update(Subject s) {
75. temp = s.getTemp();
76. city = s.getName();
77. System.out.println("Meteo attuale di " + city);
78. iconFactory = **new** IconFactory();
79. **if** (temp < 4) {
80. Icon snow = iconFactory.getIcon("SNOW");
81. snow.print();
82. System.out.println(" " + temp + "° C");
83. } **else** **if** (temp < 15) {
84. Icon cloud = iconFactory.getIcon("CLOUD");
85. cloud.print();
86. System.out.println(" " + temp + "° C");
87. } **else** **if** (temp > 14) {
88. Icon sun = iconFactory.getIcon("SUN");
89. sun.print();
90. System.out.println(" " + temp + "° C");
91. }
92. }
93. }

96. **class** IconFactory **extends** AbstractFactory {
98. @Override
99. **public** Icon getIcon(String iconType) {
101. **if** (iconType == **null**) {
102. **return** **null**;
103. }
105. **if** (iconType.equalsIgnoreCase("CLOUD")) {
106. **return** Cloud.getInstance();
108. } **else** **if** (iconType.equalsIgnoreCase("SNOW")) {
109. **return** Snow.getInstance();
111. } **else** **if** (iconType.equalsIgnoreCase("SUN")) {
112. **return** Sun.getInstance();
113. }
115. **return** **null**;
116. }
117. }

Observer.java file:

1. **package** observer;
3. **public** **interface** Observer {
4. **void** update(Subject s);
5. }

Subject.java file:

1. **package** observer;
3. **import** java.util.ArrayList;
4. **import** java.util.List;
6. **public** **abstract** **class** Subject {
7. **private** List<Observer> observers = **new** ArrayList<>();
9. **public** **void** attach(Observer o) {
10. observers.add(o);
11. }
13. **public** **void** detach(Observer o) {
14. observers.remove(o);
15. }
17. **protected** **void** notifyObservers() {
18. **for** (Observer o : observers)
19. o.update(**this**);
20. }
22. **public** **abstract** **int** getTemp();
24. **public** **abstract** **void** setTemp(**int** s);
26. **public** **abstract** String getName();
27. }

AbstractFactory.java file:

1. **package** abstractFactory;
3. **public** **abstract** **class** AbstractFactory {
4. **public** **abstract** Icon getIcon(String icon);
5. }

Icon.java file:

1. **package** abstractFactory;
3. **public** **interface** Icon {
4. **void** print();
5. }

Cloud.java file:

1. **package** singleton;
3. **import** abstractFactory.Icon;
5. **public** **class** Cloud **implements** Icon {
6. **private** Cloud() {
7. }
9. **private** **static** Cloud instance;
11. **public** **static** Cloud getInstance() {
12. **if** (instance == **null**)
13. instance = **new** Cloud();
14. **return** instance;
15. }
17. @Override
18. **public** **void** print() {
19. System.out.print("☁");
20. }
21. }

Snow.java file:

1. **package** singleton;
3. **import** abstractFactory.Icon;
5. **public** **class** Snow **implements** Icon {
6. **private** **static** Snow instance;
7. **private** **static** **final** String ANSI\_RESET = "\u001B[0m";
8. **private** **static** **final** String ANSI\_BLUE = "\u001B[34m";
10. **private** Snow() {
11. }
13. **public** **static** Snow getInstance() {
14. **if** (instance == **null**)
15. instance = **new** Snow();
16. **return** instance;
17. }
19. @Override
20. **public** **void** print() {
21. System.out.print(ANSI\_BLUE + "❄" + ANSI\_RESET);
22. }
23. }

Sun.java file:

1. **package** singleton;
3. **import** abstractFactory.Icon;
5. **public** **class** Sun **implements** Icon {
6. **private** Sun() {
7. }
9. **private** **static** Sun instance;
10. **private** **static** **final** String ANSI\_YELLOW = "\u001B[33m";
11. **private** **static** **final** String ANSI\_RESET = "\u001B[0m";
13. **public** **static** Sun getInstance() {
14. **if** (instance == **null**)
15. instance = **new** Sun();
16. **return** instance;
17. }
19. @Override
20. **public** **void** print() {
21. System.out.print(ANSI\_YELLOW + "☀" + ANSI\_RESET);
22. }
23. }

****

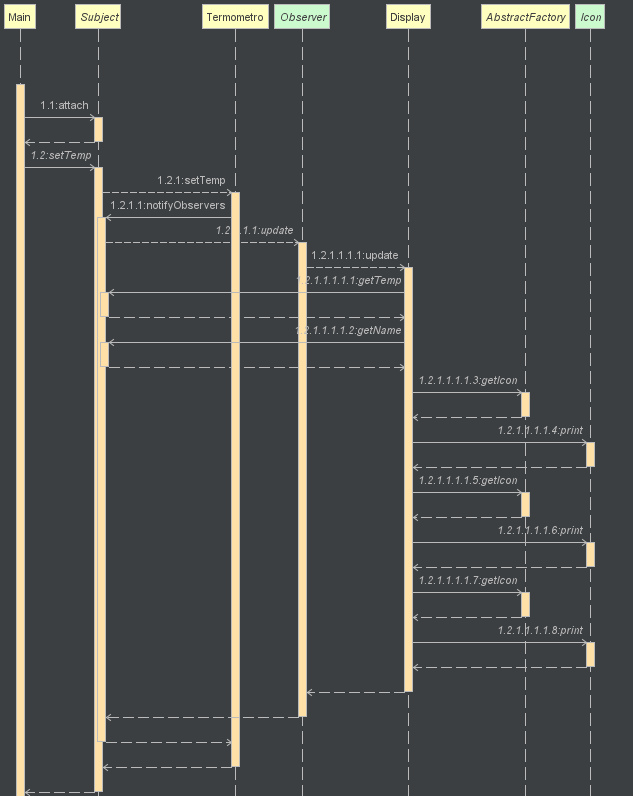
**Repository pubblico dell’elaborato:** [**https://github.com/edoardore/ElaboratoSWE**](https://github.com/edoardore/ElaboratoSWE)

**Output:**

Una porzione dell’output che risulta dalla compilazione dei moduli java.

****

**Sequence Diagram:**

****