

Elaborato Ingegneria del Software

Ilda Serjanaj

**Applicativo Java simulazione monitoraggio di un uliveto**

A.A 2019-2020

Indice

[Introduzione 3](#_Toc50489073)

[Contenuti 4](#_Toc50489074)

[Metodo 4](#_Toc50489075)

[Progettazione 6](#_Toc50489076)

[**1.1** **Casi D’uso** 6](#_Toc50489077)

[**1.2** **Class Diagram** 6](#_Toc50489078)

[Implementazione 8](#_Toc50489079)

[**2.1 Classi ed Interfacce** 8](#_Toc50489080)

[**2.1.1 Ulivo** 9](#_Toc50489081)

[**2.1.2 MallattiaPianta** 11](#_Toc50489082)

[**2.1.3 Terreno** 13](#_Toc50489083)

[**2.1.4 Campo** 13](#_Toc50489084)

[**2.1.5 Concimazione** 15](#_Toc50489085)

[**2.2** **Design Pattern** 16](#_Toc50489086)

[**2.2.1** **Observer** 16](#_Toc50489087)

[**2.2.2** **Srategy** 17](#_Toc50489088)

[Testing ed esecuzione 20](#_Toc50489089)

[**3.1 Unit Testing** 20](#_Toc50489090)

[**3.2 Sequance Diagram** 22](#_Toc50489091)

Introduzione

## Contenuti

L’elaborato, scritto in linguaggio Java, simula in maniera semplicistica la manutenzione di un Oliveto.

Attraverso il software è possibile monitorare un terreno adibito alla coltivazione degli ulivi, potendoli inserire o rimuovere tenendo conto delle dimensioni del terreno stesso.

Possono essere coltivate quattro varietà di ulivi: frantoio, leccino, moraiolo e pendolino; ognuno con le proprie caratteristiche in termini di produzione di olive e resa finale di olio.

Un parametro fondamentale per l’olivo è la decisione sul tipo di coltivazione: se biologica oppure no.

Ogni pianta è sottoposta a monitoraggio sul proprio stato di salute, rilevando vari tipi di malattie (batteriche, fungine ecc.) e a seconda della scelta del parametro di cui parlavamo precedentemente verrà scelta una strategia antiparassitaria adeguata. Quando un olivo si ammala subisce una perdita della produzione delle olive se non viene curato, altrimenti, a seconda della cura si ha sempre una diminuzione di produzione ma più lieve (per semplicità la percentuale del danno è stata impostata a 5 nel caso di utilizzo di pesticidi e a 10 nel caso di cura biologica, mentre nel momento in cui la pianta si ammala viene impostato ad un valore random compreso tra 0 e 50).

Viene effettuato anche un controllo, attraverso delle analisi chimiche del suolo, per l’elaborazione di un corretto piano di concimazione. I microelementi analizzati sono i principali, ovvero l’azoto (N), il fosforo (P) e il potassio (K), nel caso in cui risulti una carenza di uno dei tre verrà rintegrato nel suolo attraverso una concimazione mirata alla reimmisione dell’elemento carente.

## Metodo

Per la realizzazione di tale elaborato è stato utilizzato il linguaggio di programmazione Java attraverso l’IDE NeatBeans IDE 8.2.

Sono stati identificati i casi d’uso e rappresentati attraverso gli use case diagrams.

È stata inoltre definita una logica di dominio in prospettiva di specifica attraverso un class diagram in UML. Nella struttura principale del programma sono stati adottati alcuni pattern comportamentali come il pattern Observer per il monitoraggio degli olivi e degli elementi chimici del terreno, e lo Strategy per l’individuazione della cura da applicare alla pianta malata. Le varie malattie e le rispettive descrizioni e cure sono state trovate online e caricati nel programma attraverso semplici file di testo (.txt). È presente anche la realizzazione di alcune classi di testing (Unit Testing) attraverso il framework JUnit 4 integrato nell’ambiente di sviluppo.

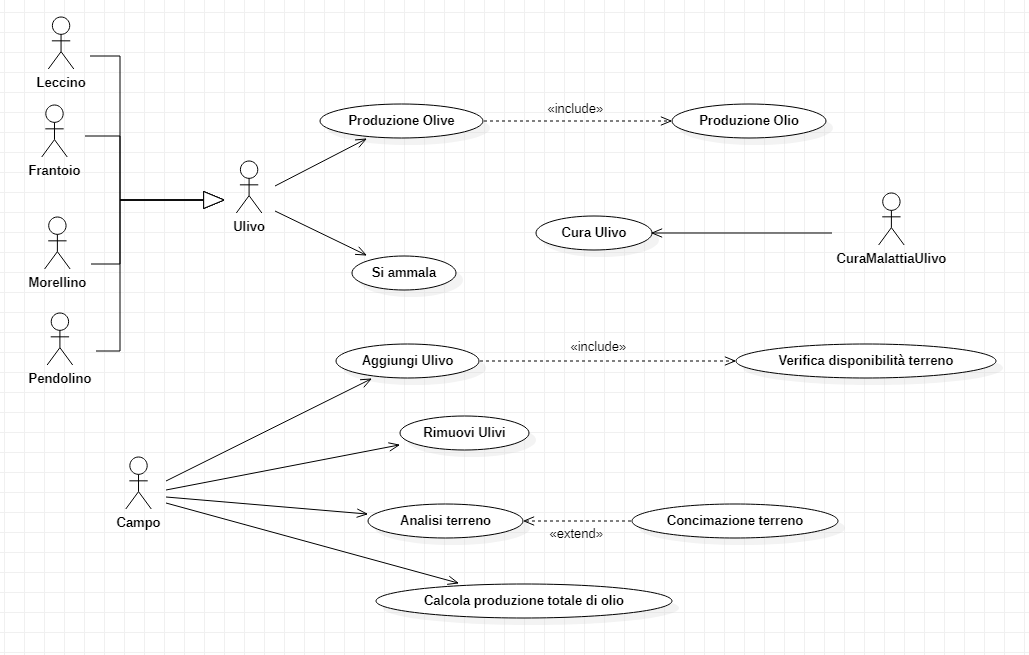
Per rendere il tutto leggermente più automatico/realistico le malattie e i vari parametri del terreno sono assegnati in modo random, come pure la produzione/ perdita di olive delle singole piante.

CAPITOLO 1

Progettazione

* 1. **Casi D’uso**

Individuazione degli attori e dei vari casi d’uso.



* 1. **Class Diagram**

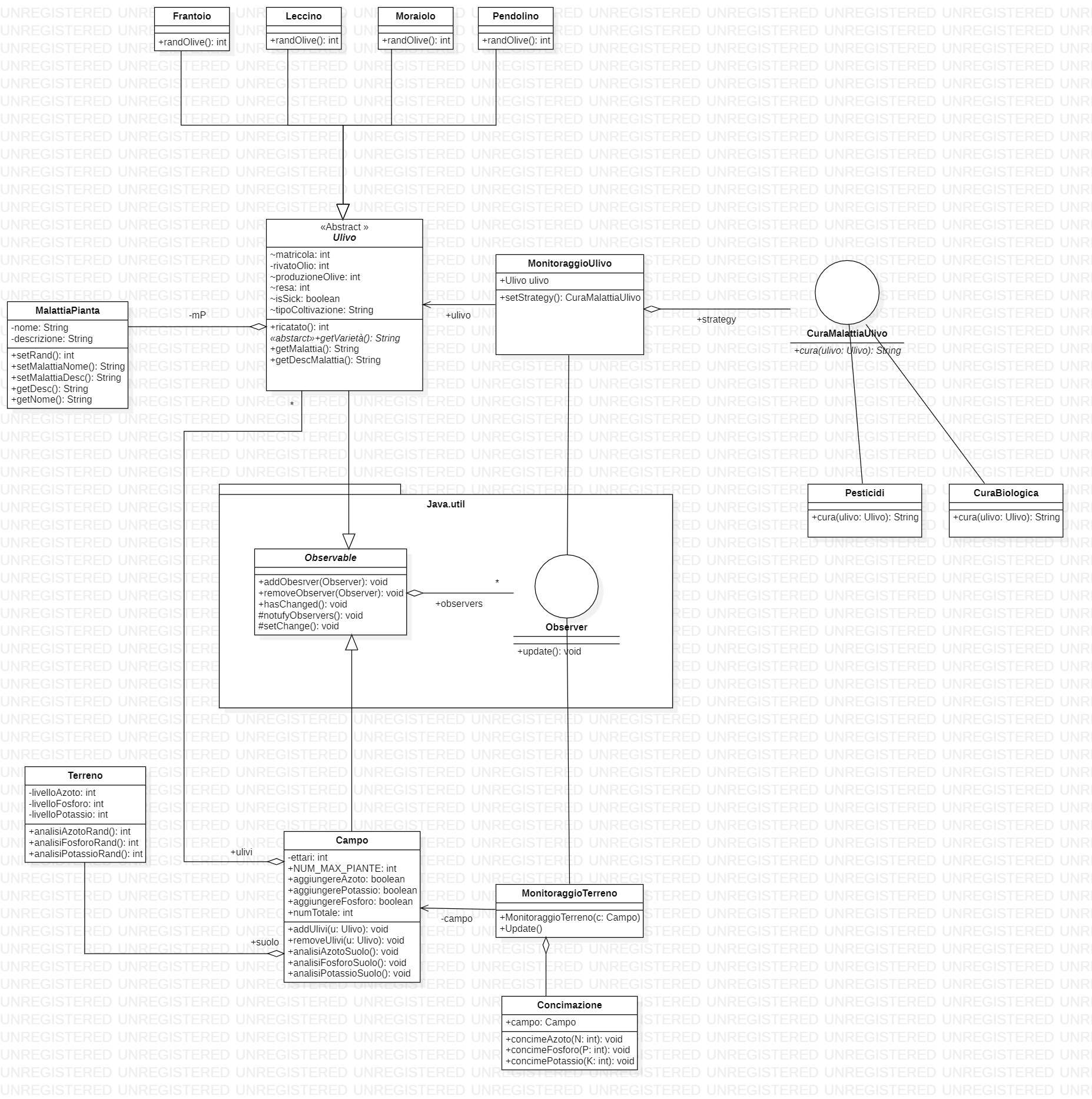
Diagramma UML che descrive la logica di dominio.

Figura 1.1 Class Diagram

CAPITOLO 2

Implementazione

**2.1 Classi ed Interfacce**

Per l’implementazione dell’applicazione sono state definite delle nuove classi ed interfacce ed utilizzate quelle della libreria standard di Java, come ad esempio quelle contenute nel package java.util.

Le principali classi dell’applicazione sono:

1. **Ulivo** (classe astratta avente quattro diverse implementazioni: **Frantoio**, **Leccino**, **Moraiolo**, **Pendolino**
2. **MalattiaPianta**
3. **MonitoraggioUlivo**
4. **CuraMalattiaUlivo** (interfaccia avente due diverse realizzazioni **CuraBiologica** e **Pesticidi**
5. **Campo**
6. **Terreno**
7. **MonitoraggioTerreno**
8. **Concimazione**

Oltre a queste classi ne è stata definita un’altra per la gestione delle eccezioni NoEttariException.

### **2.1.1 Ulivo**

La classe astratta Ulivo rappresenta la generalizzazione di una pianta di ulivo, la quale potrà essere specializzata in quattro varietà: ovvero le classi Leccino, Frantoio, Moraiolo, Pendolino.

Ho scelto di implementare questa classe come una classe astratta per beneficiare dei vantaggi dati dalla pratica della generalizzazione, e potendo implementare dei metodi comuni alle altre quattro classi specifiche (come ad esempio i metodi ulivoSiAmmala(), olioRicavato()), cosa che non sarebbe stato possibile fare se fosse stata usata un’interfaccia. La pianta può venire attaccata da diversi parassiti e ammalarsi.

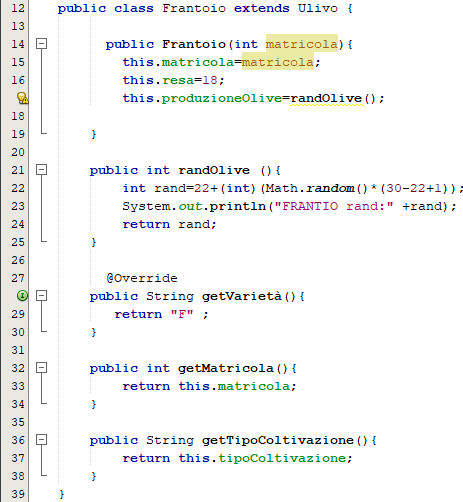
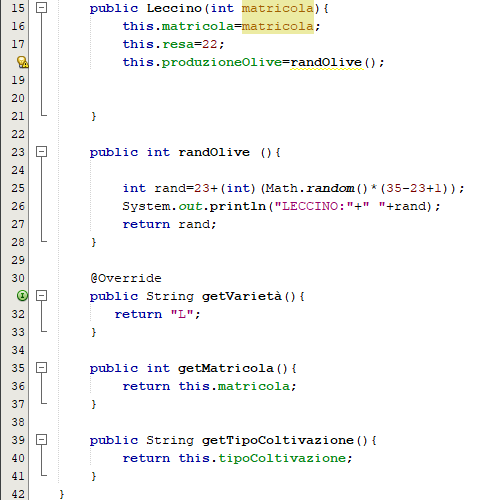


Figura .1 Frammento di codice class Ulivo

Attraverso l’attributo perditaOlive, si indica la perdita percentuale di olive in seguito alla malattia della pianta (l’attributo viene settato in modo random), nel caso in cui venga curata, a seconda del metodo antiparassitario usato, questa percentuale diminuirà.

Il metodo ulivoSiAmmala(), cambierà lo stato della pianta da sana a malata, settando l’attributo booleano isSick da false a true. Mentre il metodo ulivoGuarisce() verrà chiamato dalla giusta implementazione della classe CuraMalattiaUlivo, e a seconda della cura verrà stabilita la percentuale di perdita di olive.

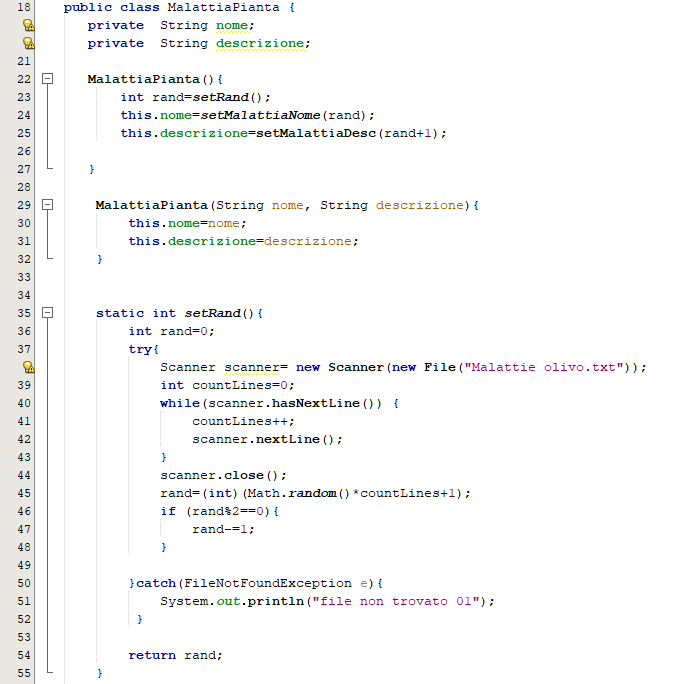
Le implementazioni di Ulivo andranno a specificare i vari attributi con i dati che appartengono alla varietà stessa.

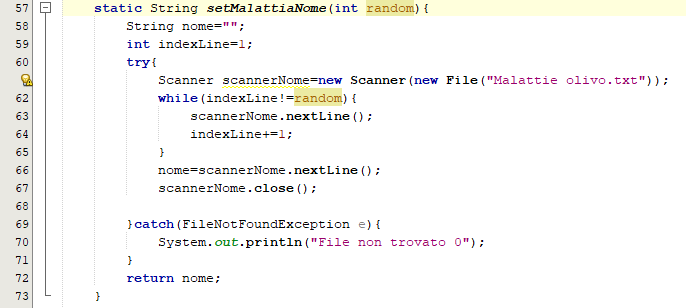


**Figura 2.2** *Frammento codice Class Leccino* **Figura 2.3** *Frammento codice Class Frantoio*

### **2.1.2 MallattiaPianta**

La classe MalattiaPianta rappresenta la malattia che colpisce l’ulivo. La malattia viene scelta random da un file .txt in cui è presente un campione di quindici malattie e le rispettive descrizioni. La classe avrà come attributi il nome della patologia e una piccola descrizione in cui verranno spiegati brevemente i danni che essa può causare alla pianta.





**Figura 2.4** *Frammenti codice Class MalattiaPianta()*

### **2.1.3 Terreno**

La classe Terreno indica il suolo del campo. Ha come attributi i vari elementi chimici che lo possono costituire. Esso viene associato ad ogni Campo.

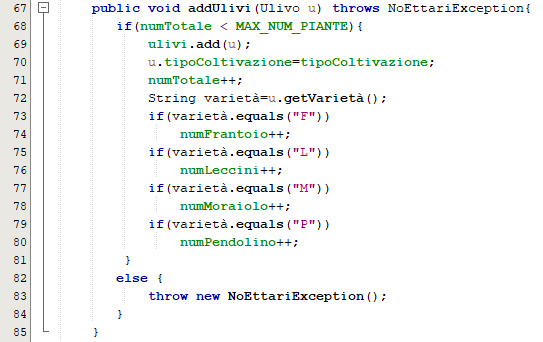


**Figura 2.5** *Frammenti Class Terreno*

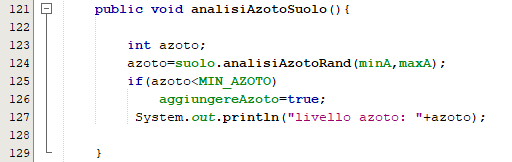
### **2.1.4 Campo**

La classe Campo rappresenta l’intero oliveto. Ogni oliveto avrà una certa dimensione espressa dall’attributo ettari. Dopodiché conterrà una lista (implementata come ArrayList) degli olivi. Ogni campo adotterà un metodo di coltura: biologico o non biologico, e a seconda di questa decisione verrà gestito il caso di in cui una pianta si ammali.

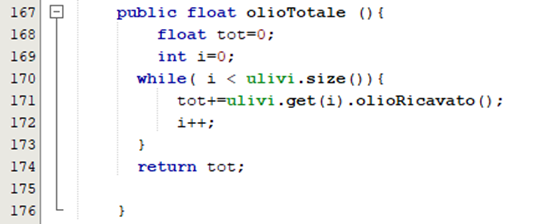
Conterrà pure un attributo di tipo Terreno, con il quale andremo ad indicare le proprietà chimiche del suolo del campo stesso, attraverso una serie di metodi atti al rilevamento di potassio, fosforo e azoto, e sempre per questo motivo sono presenti una serie di attributi che indicano i valori minimi e massimo dei vari elementi. Infine, contiene il metodo olioTotale() il quale restituisce l’olio ricavato dalle olive degli olivi del campo.



**Figura 2.6** *Metodo addUlivi()*



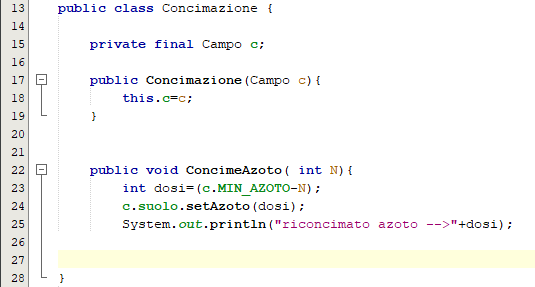
**Figura 2.7** *Metodo analisiAzotoSuolo()*

****

**Figura 2.8** *Metodo olioTotale()*

### **2.1.5 Concimazione**

Con questa classe si rappresentano i vari tipi di concimi che saranno dati al terreno nel caso in cui, dalle analisi del suolo, risulti una carenza di uno o più elementi

.

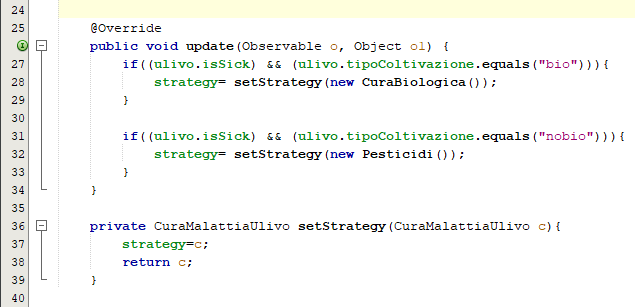
**Figura 2.9** Frammento codice Class Concimazione

* 1. **Design Pattern**

Sono stati usati alcuni design pattern per la realizzazione del progetto. In particolare i pattern utilizzati sono :

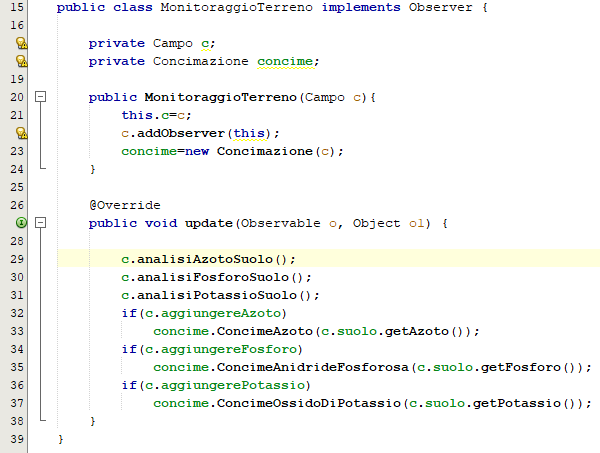
1. **Observer** (in due casi diversi)
2. **Strategy**
   * 1. **Observer**

Il pattern **Observer** è un pattern comportamentale utilizzato quando si ha la necessità di monitorare il comportamento (ad esempio attraverso il cambiamento dello stato) di uno più oggetti, detti subject, attraverso altri oggetti detti observers. Il subject ha al suo interno una lista degli observers che lo stanno “osservando”, e sarà suo compito notificare ad essi ogni volta che cambierà stato. Per l’implementazione sono state usate la classe **Observable** e l’interfaccia Observer entrambe contenute nel package **java.utill**. In particolare, nel progetto abbiamo due diversi utilizzi di questo pattern. La prima è quella in cui la classe **Ulivo** è il subject (che estenderà Observable), mentre la classe che funge da Observer è la classe **MonitoraggioUlivo**.



**Figura 2.10** Frammento Class MonitoraggioUlivo

Mentre il secondo utilizzo di Observer è per monitorare i livelli di azoto, potassio e fosforo nel suolo del campo che prendiamo in considerazione, dunque il subject questa volta sarà **Campo** (dal quale arriviamo al suo Terreno) mentre l’observer è **MonitoraggioTerreno.**



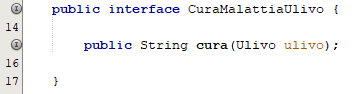
**Figura 2.11** *class MonitoraggioTerreno*

* + 1. **Strategy**

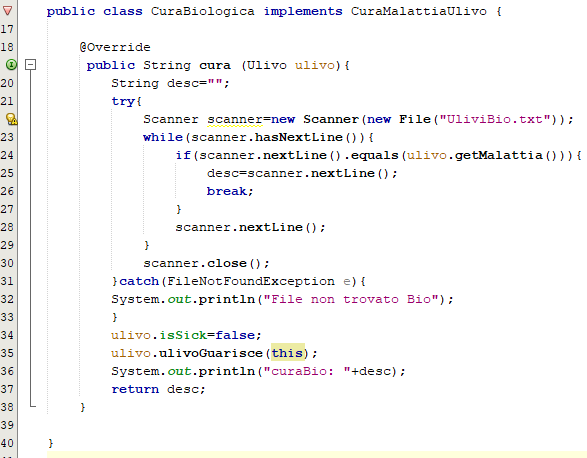
Il pattern **Strategy** è un pattern comportamentale utilizzato per avere un comportamento diverso di un determinato metodo (incapsulato in un oggetto) a seconda della strategia scelta (mediante il metodo **setStrategy()** ).

All’interno dell’applicazione l’interfaccia **CuraMalattiaUlivo** generalizza le due sottoclassi **CuraBiologica** e **Pesticidi**.

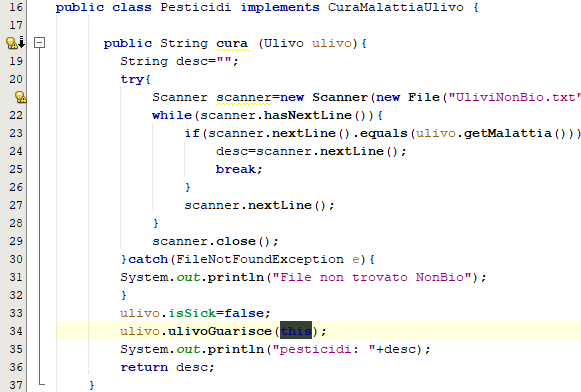
Nel nostro caso la strategia di cura nel caso una pianta si ammali cambia a seconda della scelta sul tipo di coltivazione fatto in precedenza, ovvero l’attributo **tipoColtivazione** nella classe Campo. Se impostata a “bio” la pianta verrà curata con una cura biologica (**CuraBiologica**) che nella stragrande maggioranza dei casi consiste nell’immissione di particolari specie di “antagonisti” di ciò che danneggia la pianta; altrimenti verranno usati pesticidi chimici.



**Figura 2.12** *Frammento codice interfaccia CuraMalattiaUlivo*



**Figura 2.13** *codice class CuraBiologica*



**Figura 2.14** *codice class Pesticidi*

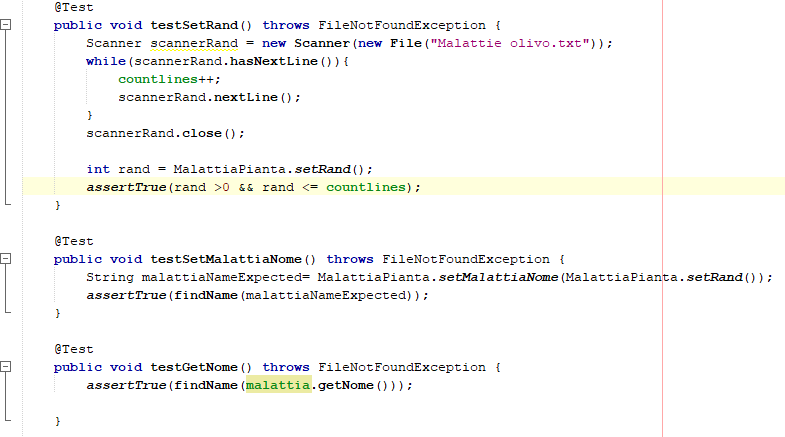
CAPITOLO 3

­Testing ed esecuzione

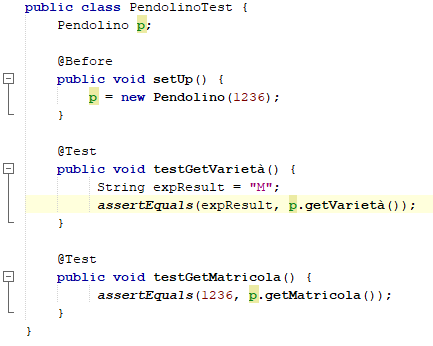
## **3.1 Unit Testing**

Con Unit Testing si intende la verifica di singole porzioni di codice, ovvero le “unità” : la parte più piccola testabile all’interno di un software. Nel caso della programmazione Object-Oriented (OOP) saranno le singole classi o i singoli metodi. Una volta individuate le varie sezioni di codice si potrà procedere con i test che vengono detti **test cases.**

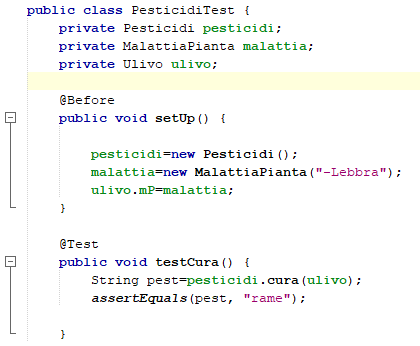
All’interno del progetto è stato usato il framework **JUnit 4.12.** Sono state testate le classi **Leccino**, **Moraiolo**, **Frantoio**, **Pendolino**, **MalattiaPianta**, **Pesticidi**, **CuraBiologica** e **Concimazione**



**Figura 3.1** *Frammento codice class MalattiaPianteTest()*

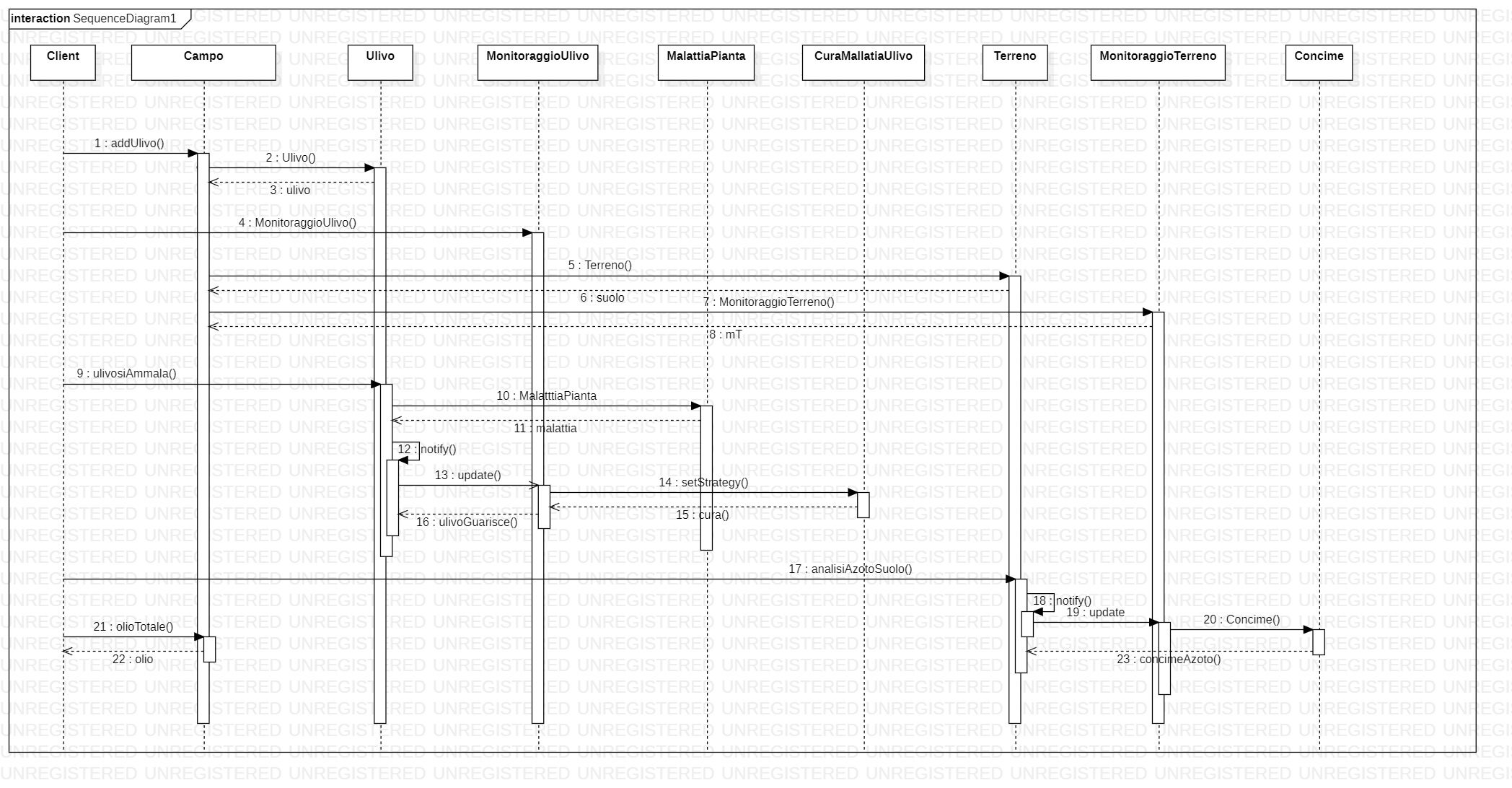


**Figura 3.2** *Frammento codice class PendolinoTest()*



***Figura 3.3*** *Frammento codice class PesticidiTest()*

## **3.2 Sequance Diagram**

Simulazione possibile flusso di esecuzione con inserimento di un unico ulivo.