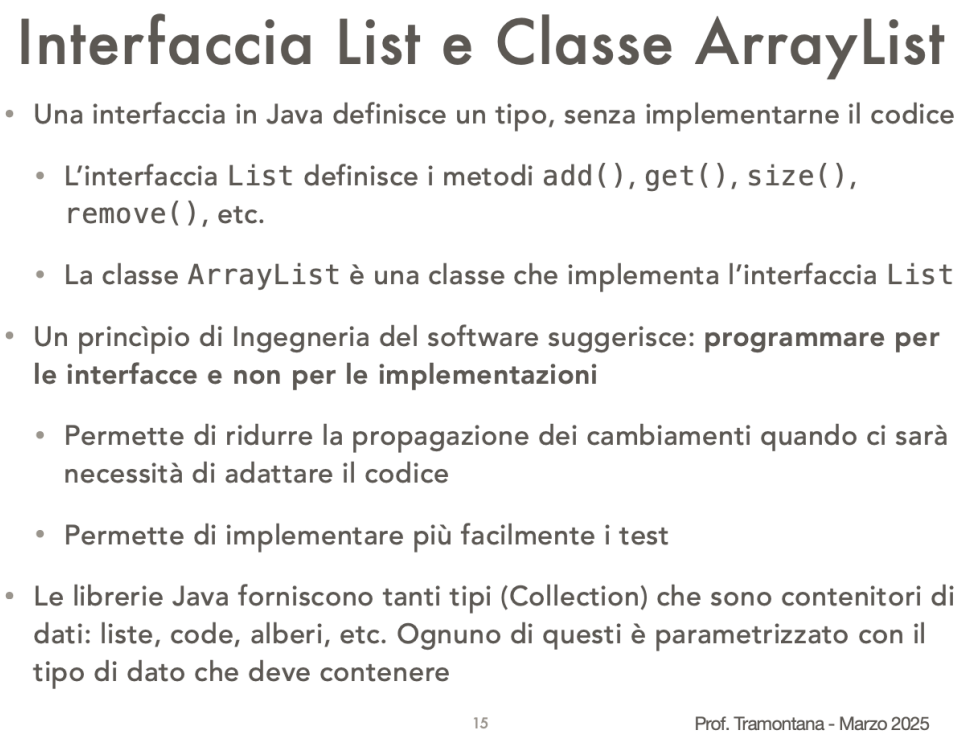
risultati, se uguali a quelli dei requisiti allora il sistema si comporta in modo corretto (Domanda esame: come verifico se un sistema software è corretto?).

Per essere corretto si deve verificare non solo se rispetta i requisiti ma anche se è corretto per il cliente.

# lezione 9 marzo

Quando sviluppiamo un sistema software di grandi dimensioni, dobbiamo conoscere bene le librerie, é sbagliato implementare codice se è già nelle librerie.

## Interfaccia




In java esiste una variante alle classi di c++ : Interfaccia.

Interface definisce un tipo, utilizzato per definire variabili ma non implementa codice.

Per implementarlo dobbiamo dare un nome al tipo e indicare quali sono i metodi pubblici dell'interfaccia e nessuno è implementato, assomiglia molto alla classe astratta in c++.

La classe ArrayList implementa i metodi dell'interfaccia List. Significa che lo spazio allocato in memoria è contiguo, quindi l'accesso in lettura è veloce, sia ordinati che non. La dimensione è fissata dalle librerie, se questo spazio si riempie, le librerie AUTOMATICAMENTE allocano maggior spazio contiguo altrove. lo spazio non si esaurisce mai.

se nel rettangolo del diagramma UML c'è la scritta <<interface>> si chiama Stereotipo con sotto List(che è interfaccia), oppure rettangolo con scritta List dentro con sopra a destra un cerchio, oppure ancora un cerchio con un nome TUTTE PER INDICARE INTERFACCIA(mettere foto)

List definisce il comportamento di una lista, a livello di nomi di metodi che sono utili per i meccanismi fondamentali in uso della lista(aggiungere elementi, togliere elementi...), e possono essere implementati in vari modi(con array, liste concatenate). si comporta come contenitore.

Non devo implementarla io ma mi appoggio alle libreria java.

Puo contenere sia oggetti che tipi primitivi. Se voglio dire quale elemento va dentro la lista, devo inserire anche il parametro che indica il tipo degli elementi contenuta nella lista, es. List<String>.

```
private List<String> l;
```

così, dichiaro la lista nella classe HelloWorld, e aggiungo

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

per usare la libreria.

```
import java.time.LocalDate;
import java.util.List;
```

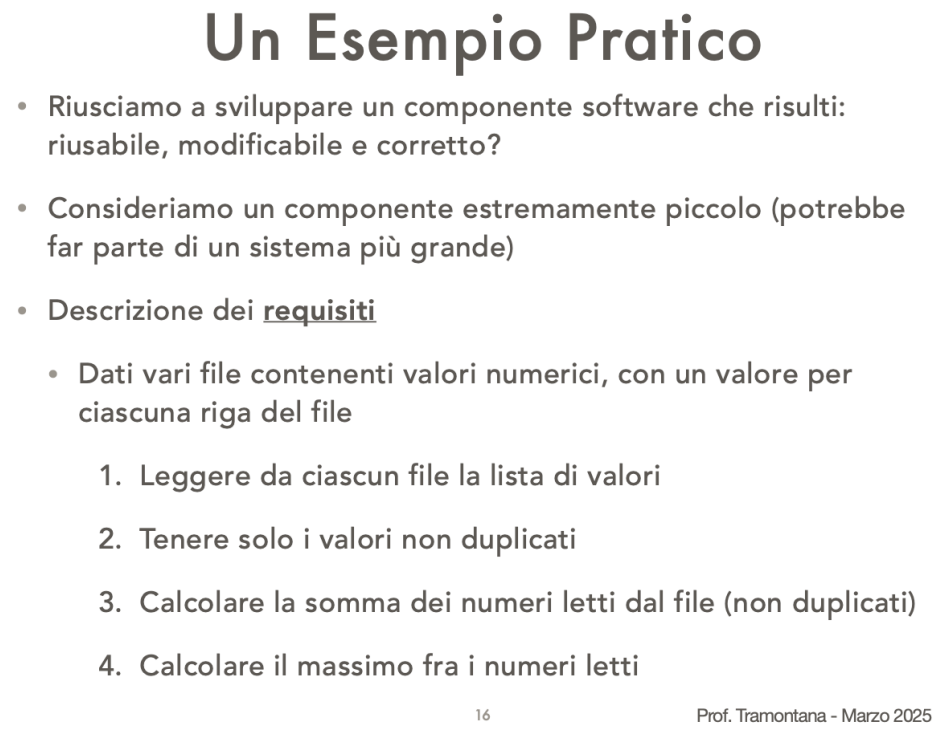
```
public class HelloWorld {
    private static int valore = 0;
    private int contatore;
    private String saluto="Ciao";
    private static LocalDate data= LocalDate.now();
    private List<String> l;

    private void aggiorna(){
        lista = new ArrayList<>();// cosi alloco
        lista.add("Ciao");//aggiungo elementi
        lista.add("Bye bye");
        /*
        for(int i=0; i< lista.size(); i++){
                String s =lista.get(i);
                System.out.println(s);
        }
        */
        for( String s: lista){//ciclo for avanzato
            System.out.println(s);
        }
      }

    public static void main(String[] args){
        System.out.println("Hello World.");

        HelloWorld h=new HelloWorld();
        h.stampe();
    }

    private void stampe(){
        valore++;
        System.out.println(data);
        IO.println("ciao ciao");
        IO.println("valore : " +valore);
        System.out.println(saluto + ", Ingegneria del softwar
e.");
```

```
        contatore++;
        System.out.println("contatore: " +contatore);
    }
}
```

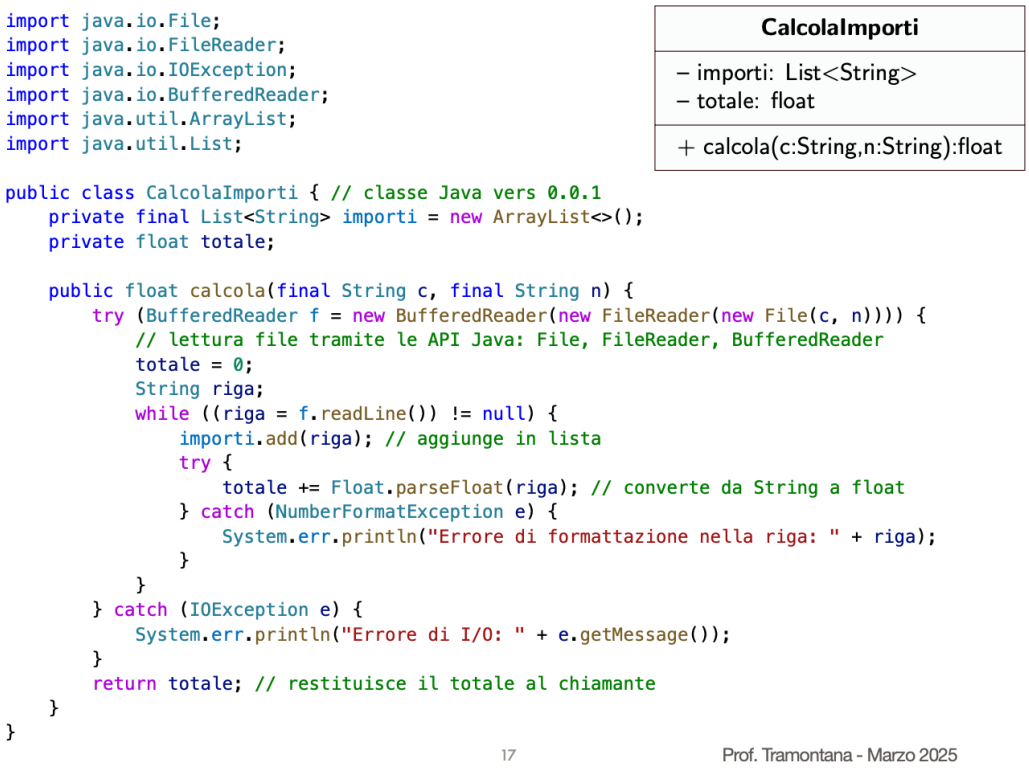
```
 for( String s: lista){//ciclo for avanzato
           System.out.println(s);
       }
```

tipo, nome , dentro lista, s prenderà gli elementi dentro la lista e li stamperà. LO uso quando non mi serve l'indice, altrimenti uso l'altro.

## Esempio pratico



Se dobbiamo leggere elementi presenti in un file e dobbiamo poi sommarli, vedere il massimo e quant'altro.  Prima di scrivere codice, capisco quali sono gli obiettivi e scelgo cosa usare in base a questo. Avere una traccia scritta di cosa dovrò fare. dobbiamo capire se il codice è corretto, modificarlo, dire che è modificabile, usabile, mantenibile ecc




```java
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
import java.util.ArrayList;
import java.util.List;

public class CalcolaImporti { // classe Java vers 0.0.1
    private final List<String> importi = new ArrayList<>();
    private float totale;

    public float calcola(final String c, final String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            // lettura file tramite le API Java: File, FileReader, BufferedReader
            totale = 0;
            String riga;
            while ((riga = f.readLine()) != null) {
                importi.add(riga); // aggiunge in lista
                try {
                    totale += Float.parseFloat(riga); // converte da String a float
                } catch (NumberFormatException e) {
                    System.err.println("Errore di formattazione nella riga: " + riga);
                }
            }
        } catch (IOException e) {
            System.err.println("Errore di I/O: " + e.getMessage());
        }
        return totale; // restituisce il totale al chiamante
    }
}
```

17 Prof. Tramontana - Marzo 2025

Qui è bene usare una lista, non sappiamo quanti sono gli elementi.

# Librerie Java

- Riepilogo di alcuni tipi e metodi di librerie Java utilizzati
- `List`, interfaccia utile a tenere una sequenza di elementi
- `ArrayList`, implementazione di `List`, la sua dimensione cresce automaticamente
- `add()`, metodo di `List`, aggiunge un elemento alla fine della lista
- `contains()`, metodo di `List`, ritorna true se la lista contiene l'elemento specificato
- `parseFloat(String s)`, metodo di `Float`, ritorna un nuovo float con il valore specificato nel parametro stringa s, o ritorna un'eccezione se la stringa non contiene un numero
- `readLine()`, metodo di `LineNumberReader`, ritorna una stringa contenente la linea del file, o `null` se si raggiunge la fine del file

# Linguaggio Java

- Parole chiave
  - **float** si usa per dichiarare una variabile che può tenere numeri in virgola mobile; si usa pure per dichiarare che un metodo restituisce un valore float
  - **if** si usa per creare un'istruzione condizionale
  - **return** si usa per concludere l'esecuzione di un metodo, se seguita da un valore quest'ultimo è restituito al chiamante
  - **try** si usa quando la chiamata di metodi potrebbe lanciare una eccezione, che si cattura con **catch**
  - **while** si usa per creare un ciclo

18 Prof. Tramontana - Marzo 2025

List<String> importi, uso String come tipo poichè contiene file semplici(?), per cose piu complicate devo avere altre librerie di supporto.

il metodo calcola inizia a fare il lavoro necessario segnato nella lista dei requisiti.

per leggere un file possiamo usare:

```
public float calcola(final String c, final String n) {
    try (BufferedReader f = new BufferedReader(new FileReader
(new File(c, n)))) {
 // lettura file tramite le API Java: File, FileReader, Buffe
redReader
```

c ed n sono due parametri che indicano cartella e nome file; vengono utilizzate dalla riga sottostante.

File in java è il tipo per il file, e passo la cartella come String(percorso o relativo quindi il punto in cui sono, o assoluto) e il nome del file.

la classe FileReader la istanzio pure e la uso per leggere il file che istanzio con tipo File. potrei loeggere carattere per carattere SOLO con FileReader, se voglio leggere intere righe uso BufferedReader, fino all'eof. f è l'assegnazione di tutto ciò.

blocco try-catch, gestione delle eccezioni(cioe che trova errore, qualcosa che non posso completare): try cerca di fare tutto qwueooo che dico, se lo fa va avanti, se lancia un'eccezione le istruzioni di seguito non vengono eseguite e quindi invia il messaggio di errore

qui l'istruzione che potrebbe lanciare l'eccezione è tra parentesi tonde, oppure uso direttamente le graffe.

```
totale = 0;
String riga;
while ((riga = f.readLine()) != null) {
    importi.add(riga); // aggiunge in lista
```

f.readLine() permette di leggere questa linea e sposta il puntatore per farlo.

```
try {
        totale += Float.parseFloat(riga); // converte da Stri
ng a float
    } catch (NumberFormatException e) {
```

passo al metodo parseFloat la stringa per convertire string a float, se non ce qualche carattere alfabetico o numerico lancia l'eccezione a runtime:

```
catch (NumberFormatException e) {
    System.err.println("Errore di formattazione nella riga: "
+ riga);
}
```

```
} catch (IOException e) {
System.err.println("Errore di I/O: " + e.getMessage());
}
return totale;
```

qui il catch finale.

## Ulteriore implementazione

```java
public class CalcolaImporti { // classe Java vers 0.0.2
    private final List<String> importi = new ArrayList<>();
    private float totale, massimo;

    public float calcola(final String c, final String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            totale = massimo = 0;
            String riga;
            while ((riga = f.readLine()) != null) {
                importi.add(riga); // aggiunge in lista
                try {
                    float valore = Float.parseFloat(riga);
                    totale += valore;
                    if (valore > massimo)
                        massimo = valore; // aggiorna massimo se necessario
                } catch (NumberFormatException e) {
                    System.err.println("Errore di formattazione nella riga: " + riga);
                }
            }
            f.close();
        } catch (IOException e) {
            System.err.println("Errore di I/O: " + e.getMessage());
        }
        return totale; // restituisce il totale al chiamante
    }
}
```

21 Prof. Tramontana - Marzo 2025

per individuare il massimo;

```java
public class CalcolaImporti { // classe Java vers 0.1
   private final List<String> importi = new ArrayList<>();
   private float totale, massimo;

   public float calcola(final String c, final String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            totale = massimo = 0;
            String riga;
            while ((riga = f.readLine()) != null) {
                if (!importi.contains(riga)) {
                    importi.add(riga);
                    try {
                        float valore = Float.parseFloat(riga);
                        totale += valore;
                        if (valore > massimo)
                            massimo = valore;
                    } catch (NumberFormatException e) {
                        System.err.println("Errore di formattazione in: " + riga);
                    }
                }
            }
            f.close();
        } catch (IOException e) {
            System.err.println("Errore di I/O: " + e.getMessage());
        }
        return totale;
    }
}
```



contains utile per verificare se l'argomento che passo è già dentro la lista.

In questo modo ho completato i requisiti richiesti.

Posso aggiugere qui la chiamata a calcola:

```java
import java.time.LocalDate;

public class HelloWorld {
    private static int valore = 0;
    private int contatore;
    private String saluto="Ciao";
    private static LocalDate data= LocalDate.now();

    public static void main(String[] args){
        System.out.println("Hello World.");
```

```
        HelloWorld h=new HelloWorld();
        h.stampe();
    }

    private void stampe(){
        valore++;
        System.out.println(data);
        IO.println("ciao ciao");
        IO.println("valore : " +valore);
        System.out.println(saluto + ", Ingegneria del softwar
e.");
        contatore++;

    }
}
```

da sistemare!!!!

## Altra versione di Calcola

```java
public class CalcolaImporti02 { // classe versione 0.2
   private List<String> importi = new ArrayList<>();
   private float totale, massimo;
   public float calcola(String c, String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            String riga;
            while ((riga = f.readLine()) != null)
                if (!importi.contains(riga)) importi.add(riga);
            f.close();
        } catch (IOException e) { System.err.println("Errore di I/O: " + e.getMessage()); }
        // calcolo del totale
        totale = 0;
        for (int i = 0; i < importi.size(); i++) {
            try {  totale += Float.parseFloat(importi.get(i));
            } catch (NumberFormatException e) {
                System.err.println("Errore di formattazione in: " + importi.get(i));
            }
        }
        // estrazione del massimo
        massimo = Float.parseFloat(importi.get(0));
        for (int i = 1; i < importi.size(); i++) {
            try {
                if (massimo < Float.parseFloat(importi.get(i)))
                    massimo = Float.parseFloat(importi.get(i));
            } catch (NumberFormatException e) {
                System.err.println("Errore di formattazione in: " + importi.get(i));
            }
        }
        return totale;
    }
}
```



ma queste due versioni, non vanno bene!

# Spaghetti Code

# Problemi?

- Il metodo `calcola` di entrambe le versioni è **spaghetti code** (un **antipattern**)
- Il codice è **monolitico**: fa troppe cose in un unico flusso. Non è un codice Object-Oriented. Conseguenze: non si può riusare, né verificarne la correttezza
  1. Come verificare che tutti i valori del file siano stati letti? Si dovrà modificare il metodo. Non è una soluzione, si dovrebbe poter verificare il comportamento del metodo dall'esterno
  2. Analogamente per verificare il calcolo di somma e totale, in più punti si dovrebbero aggiungere alcuni controlli
  3. Non si riesce a modificare facilmente o riusare il codice. Per es. se si volessero conservare tutti i valori letti, quali ulteriori effetti provoca la modifica?
- Quindi: difficoltà di comprensione e modifiche che coinvolgono varie operazioni

26 Prof. Tramontana - Marzo 2025

Quelli precedenti erano esempi di "spaghetti code", sono difficili da ragionarci su. Implementa tante cose nello stesso flusso. E' un codice che realizzo in maniera sequenziale che fa tante cose, e ciò non va bene. Il metodo deve implementare piccole funzionalità, non tutto in un unico flusso. Questo implica anche che ho codice molto più complesso: se ho un problema nel metodo che calcola massimo, allora guardo solo quello e probabilmente l'errore lo trovo facilmente; se non faccio così è una gran fatica capire dove sta l'errore.

L'obiettivo del test è capire se è corretto invocandolo, non modificandolo


ereditarietà tipica nello spaghetti code. Nella programmazione ad oggetti, una raccomandazione è che la classe deve rappresentare una astrazione, devo pensare ad un sostantivo non ad un verbo per dare il nome alla classe, es calcolaImporti è un verbo, non va bene come nome della classe.

## Esempi codice sbagliato

```java
public class CalcolaImporti02 { // classe versione 0.2
   private List<String> importi = new ArrayList<>();
   private float totale, massimo;
   public float calcola(String c, String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            String riga;
            while ((riga = f.readLine()) != null)
                if (!importi.contains(riga)) importi.add(riga);
            f.close();
        } catch (IOException e) { System.err.println("Errore di I/O: " + e.getMessage()); }
        // calcolo del totale
        totale = 0;
        for (int i = 0; i < importi.size(); i++) {
            try {  totale += Float.parseFloat(importi.get(i));
            } catch (NumberFormatException e) {
                System.err.println("Errore di formattazione in: " + importi.get(i));
            }
        }
        // estrazione del massimo
        massimo = Float.parseFloat(importi.get(0));
        for (int i = 1; i < importi.size(); i++) {
            try {
                if (massimo < Float.parseFloat(importi.get(i)))
                    massimo = Float.parseFloat(importi.get(i));
            } catch (NumberFormatException e) {
                System.err.println("Errore di formattazione in: " + importi.get(i));
            }
        }
        return totale;
    }
}
```

leggiFile()

inserisci()

calcolaSomma()

calcolaMassimo()

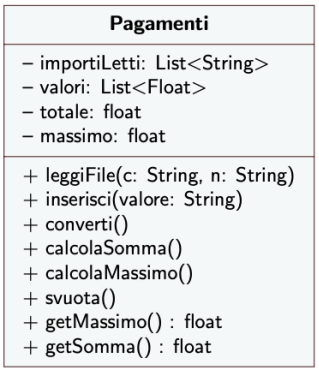
getSomma()

```java
public class CalcolaImporti { // classe Java vers 0.1
   private final List<String> importi = new ArrayList<>();
   private float totale, massimo;

   public float calcola(final String c, final String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            totale = massimo = 0;                                          leggiFile()
            String riga;
            while ((riga = f.readLine()) != null) {
                if (!importi.contains(riga)) {                             inserisci()
                    importi.add(riga);
                    try {
                        float valore = Float.parseFloat(riga);           calcolaSomma()
                        totale += valore;
                        if (valore > massimo)                         calcolaMassimo()
                            massimo = valore;
                    } catch (NumberFormatException e) {
                        System.err.println("Errore di formattazione in: " + riga);
                    }
                }
            }
            f.close();
        } catch (IOException e) {
            System.err.println("Errore di I/O: " + e.getMessage());
        }
        return totale;                                                      getSomma()
    }
}
```



# Codice non Spaghetti Code

```java
public class Pagamenti {
    private List<String> importiLetti = new ArrayList<>();
    private List<Float> valori = new ArrayList<>();
    private float totale;
    private float massimo;

    public void leggiFile(String c, String n) {
        try (BufferedReader f = new BufferedReader(new FileReader(new File(c, n)))) {
            String riga;
            while ((riga = f.readLine()) != null) inserisci(riga);
            f.close();
        } catch (IOException e) {
            System.err.println("Errore di I/O: " + e.getMessage());
        }
    }
    public void inserisci(String valore) {
        if (!importiLetti.contains(valore)) importiLetti.add(valore);
    }
    public void converti() {
        for (String importo : importiLetti)
            try {
                valori.add(Float.parseFloat(importo));
            } catch (NumberFormatException e) {
                System.err.println("Errore nella conversione di: " + importo);
            }
    }
    public void calcolaSomma() {
        totale = 0;
        for (float v : valori) totale += v;
    }
    public void calcolaMassimo() {
        if (!valori.isEmpty()) {
            massimo = valori.get(0);
            for (float num : valori)
                if (massimo < num) massimo = num;
        }
    }
```




```java
Pagamenti pagam = new Pagamenti();
pagam.svuota();
pagam.leggiFile("./", "Importi.csv");
pagam.converti();
pagam.calcolaSomma();
pagam.calcolaMassimo();
```



nome della classe è un sostantivo, nome dei metodi è un verbo, fare dei metodi piccoli(oltre 15 linee di codice magari posso iniziare a pensare se dividere in altri metodi). Dall'esterno decido il flusso di metodi da eseguire

## Paradigma Command and Query

# Considerazioni Sul Codice

- Si sta usando bene il paradigma di programmazione ad Oggetti (OOP)
  - Ogni metodo ha una sola piccola responsabilità
  - Il flusso di chiamate ai metodi è indipendente dai singoli algoritmi
  - Posso riusare (richiamandoli) i servizi offerti dai metodi
- Inoltre, sto usando il **paradigma Command e Query**
  - I metodi Query restituiscono un risultato (si vede dal parametro di ritorno), e non modificano lo stato del sistema
  - I metodi Command (o modificatori) cambiano lo stato del sistema ma non restituiscono un valore
  - I metodi query si possono chiamare liberamente, senza preoccupazioni sulla modifica dello stato, mentre si deve stare più attenti quando si chiamano i metodi command
- *Enhanced for* indica che si vogliono gli elementi della lista, uno per ogni passata, si può usare con i tipi che implementano `Iterable`



Command and Query(sono libero di scegliere se usarlo o no nella OOP)

command = metodi che fanno operazioni e l'obiettivo e cambiare lo stato ma non restituiscono un risultato

query = su quella classe si possono fare delle interrogazioni, e restituisce qualcosa

l metodi query osno leggeri da eseguire poichè è semplicemmente la lettyra di un valore e non aggiungo costi di elaborazione, non c'è spreco di risorse

l metodi command sono quelli che fanno lavoro, hanno cicli e variabili temporanee quindi quando le invoco spreco tanto, quindi le uso solo quando strettamente necessario.

le distinguo in base al ritorno, se c'è o no.

# Metriche

- Classe CalcolaImporti (vers. 0.1)
  - Metodi 1, LOC 18 (solo le linee che terminano con un “;”)
- Classe CalcolaImporti (vers. 0.2)
  - Metodi 1, LOC 23
- Classe Pagamenti (vers 1.1)
  - Metodi 8, LOC 27 (media 3.4 LOC per metodo)
- Confronto con sistemi software open source (valori approssimativi) JUnit (JU), JHotDraw (JHD):
  - JU LOC 22K, Classi 231, Metodi 1200, Attributi 265, media 18
  - JHD LOC 28K, Classi 600, Metodi 4814, Attributi 1151, media 6



il numero di linee di codice non è fondamentale, si preferisce avere programmazione modulare e più linee. utilizzo pattern per contare le linee di codice, ad esempio conto solo linee che terminano con ;

## Conclusione L1

# Conclusioni

- **Key points**
  - Correttezza del codice: test
  - Antipattern Spaghetti Code
  - Ciascun metodo ha un unico compito
- Esempi di domande d'esame
  - Implementare un frammento di codice che usa l'enhanced for
  - Dire come si può controllare se un codice è corretto
  - Implementare un metodo query
  - Dire qual è la differenza fra List ed ArrayList
  - Dire a cosa serve il metodo contains di List



# lezione 13 marzo

## Miglioramenti Sul Codice

1. Separazione delle varie operazioni, ciascuna in un suo metodo. Ciò permette di **verificare la correttezza delle operazioni e di riusarle**
2. **Costruzione di astrazioni**, via via più utili, ovvero metodi di livello più basso e metodi di livello più alto, che richiamano i precedenti, lo stesso si fa per le classi
3. **Separazione delle operazioni** che aggiornano lo stato (command) da quelle che restituiscono valori (query), in metodi diversi
   - **Query** sono operazioni che ottengono informazioni (accedendo a dati o facendo calcoli sui dati)
   - **Command** sono operazioni che effettuano cambiamenti
4. Lo stato dell'oggetto diventa **osservabile**, attraverso metodi opportuni, in modo da poter fare i test
   - **Test: programma che esegue il codice sotto certe condizioni (valori in input) e valuta se il risultato fornito è corretto**