Ingegneria Del Software

Prof. E. Tramontana

Fonti 📚

- Materiale e link utili
  - https://www.dmi.unict.it/tramonta/se
  - https://github.com/e-tramontana
- Gruppi Teams e Telegram per avvisi

Lezioni

- Coprono tutto il programma del corso
- Partecipazione **fortemente consigliata**: si impara di più, e si ascolta da un esperto, è possibile fare domande ed ottenere risposte
- Orario di ricevimento online (su Teams) o in presenza: mandare un messaggio su Teams per un appuntamento
- Per rendere efficace lo studio: **esercitarsi con il codice**, usare i concetti spiegati e i tool consigliati, partecipare alle lezioni
- **Modalità Esami**
  - Scritto + Orale. Scritto: test a risposte multiple, esercizi a risposta aperta che richiedono di implementare codice, e disegnare alcuni diagrammi UML
  - Progetto opzionale, da concordare (a partire da maggio)

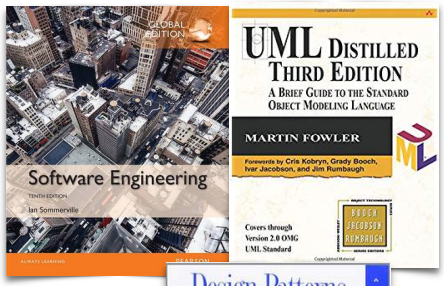
Libri Consigliati

Le slide non bastano :-(

- Sommerville. Ingegneria del Software. Pearson

<u>oppure</u>

- Pressman. Principi di Ingegneria del Software. McGraw-Hill

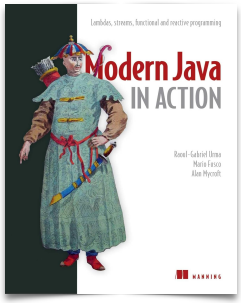


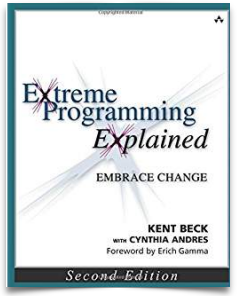
- Fowler. UML Distilled. Pearson

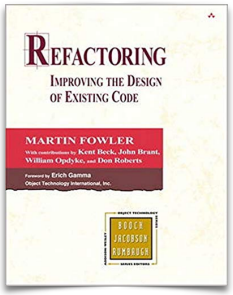
- Gamma, Helm, Johnson, Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley

Libri Consigliati

- Urma, Fusco, Mycroft. Modern Java in Action. Manning
- Beck. Extreme Programming Explained. Addison-Wesley
- Fowler. Refactoring: Improving the design of existing code. Addison-Wesley

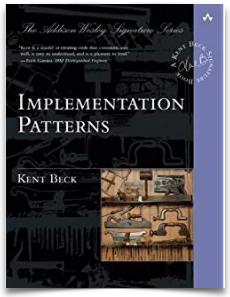


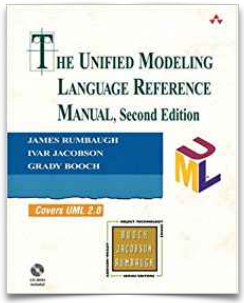




Libri Per Approfondimenti

- Beck. Implementation Patterns. Addison-Wesley
- Rumbaugh, Jacobson, Booch. The Unified Modeling Language Reference Manual. Addison-Wesley





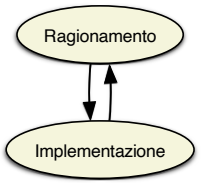
Obiettivi Del Corso

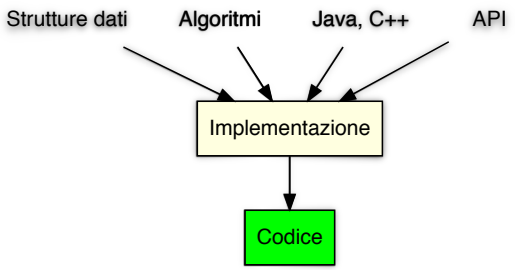
- Descrivere le pratiche utili a sviluppare un sistema software di **grandi dimensioni**, che deve andare in produzione
- Fasi dei processi di sviluppo del software: analisi (requisiti), progettazione (**OOP**, **Design Pattern**, **Refactoring**), implementazione, test (convalida), manutenzione
- Processi di sviluppo: cascata, agili (XP), etc.
- Ci si baserà sulla progettazione orientata agli oggetti (OOP)
- Si useranno: lo standard UML, il **linguaggio Java**
  - Java è attualmente molto diffuso e richiesto. Secondo vari indici (Marzo 2025), Java è: terzo su Tiobe index (dopo Python, C++), secondo su Pypl index (dopo Python), terzo su GitHub (dopo JavaScript e Python)

Perché Ingegneria Del Software?

- Perché è importante conoscere i concetti e le tecniche di Ingegneria del Software nell'era dell'AI? La gen-AI (ChatGPT, Copilot) non basta?
  - E' importante capire il codice generato poiché potrebbe contenere difetti o problemi di sicurezza; quindi bisogna saper fare debug, diagnosticare e risolvere un problema
  - Quando si sviluppa un sistema software si mette in pratica un approccio di risoluzione dei problemi, l'AI potrebbe non trovare il modo di risolvere un problema nuovo
  - L'AI non progetta l'intero sistema, ma riusa soluzioni (e algoritmi)
  - L'AI potrebbe non riuscire a inventare nuovi algoritmi, a trasformare nuove idee in sistemi software
  - Conoscere l'ingegnera del software permette di fornire prompt più precisi e a ottenere risultati migliori dall'AI

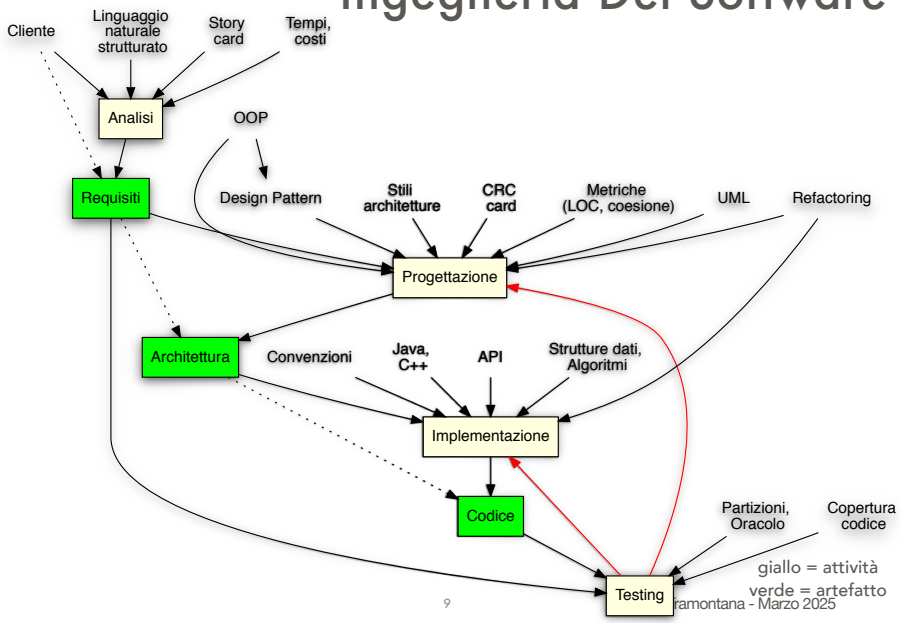
Sviluppo ...





giallo = attività
verde = artefatto

Ingegneria Del Software



Hello World In Java

```java
import java.time.LocalDate;

/**
 * Classe che stampa sullo schermo un messaggio e la data corrente
 */
public class HelloWorld { // definizione classe
    // dichiarazione e assegnazione campi
    private static final String msg = "Lezione di Ingegneria del Software";
    private static final LocalDate d = LocalDate.now();

    /**
     * Metodo da cui inizia l'esecuzione del programma
     *
     * @param args parametri passati al metodo all'avvio della classe
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
        System.out.println(msg);
        System.out.println(d);
    }
}
```

Output

```
Hello World
Lezione di Ingegneria del Software
2025-03-03
```

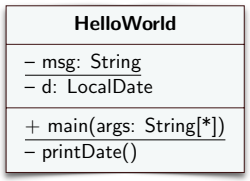


```java
import java.time.LocalDate; // indica dove trovare la classe LocalDate

public class HelloWorld { // dichiara classe HelloWorld
    private static final String msg = "Lezione di Ingegneria del Software";
    private LocalDate d; // dichiara campo d di tipo LocalDate

    public static void main(String[] args) {
        System.out.println("Hello World"); // scrive su schermo
        System.out.println(msg);
        final HelloWorld world = new HelloWorld(); // crea oggetto
        world.printDate(); // chiama metodo
    }

    private void printDate() { // metodo
        d = LocalDate.now(); // chiama metodo static now
        System.out.println(d);
    }
}
```



- Il codice della classe HelloWorld deve essere salvato sul file `HelloWorld.java`
- Compilazione: `javac HelloWorld.java`
- Esecuzione: `java HelloWorld`

Parole Chiave Di Java

- **`class`** permette di definire un tipo, e quindi le sue istanze
- **`final`** definisce un campo o una variabile che non può essere assegnata più di una volta (una costante). Una classe final non può essere ereditata, un metodo final non può essere ridefinito (override)
- **`import`** indica dove trovare la definizione di una classe che sarà usata nel seguito
- **`new`** permette di creare un'istanza di una classe
- **`private`** e **`public`** indicano l'accessibilità di classi, campi e metodi
- **`static`** è usata per dichiarare un campo o un metodo appartenente alla classe (e non all'istanza)
- **`void`** indica che il metodo non ritorna alcun valore
- Tipi usati: **`String`**, per rappresentare insiemi di caratteri; **`LocalDate`**, per accedere alla data attuale; **`System`** per scrivere sullo schermo

Caratteristiche Del Software

- **Modificabilità**: un sistema software è intrinsecamente modificabile, poiché non ha parti fisiche (non è costituito da atomi)
- Se un sistema software è di successo vi è necessità di cambiarlo
  - Per adattarlo ad una realtà che cambia (mutate esigenze)
- Le richieste di estensione aumentano al crescere del successo
- Poiché di successo, il sistema software sopravvive all'hardware per cui era stato sviluppato inizialmente, generando una nuova esigenza di adattamento alla nuova piattaforma

Qualità Del Software

- Le tecniche dell'ingegneria cercano di produrre sistemi software entro i costi e i tempi preventivati e con qualità accettabile
- Criteri operativi per valutare la qualità
  - **Correttezza**: il sistema software aderisce allo scopo ed è **conforme alle specifiche (requisiti)**
    - Il sistema software fa quello che il cliente vuole?
    - Il sistema software soddisfa le specifiche che erano state raccolte? [vedi Testing]
  - **Efficienza**: nessuno spreco di risorse
  - **Manutenibilità**: facilità a introdurre modifiche
  - **Dependability**: sicurezza (security, difesa dei dati; e safety, difesa del sistema e delle persone) e affidabilità (reliability, probabilità di operare senza guasti)
  - **Usabilità**: capacità di fornire all'utente condizioni di uso efficaci

Interfaccia List e Classe ArrayList

- Una interfaccia in Java definisce un tipo, senza implementarne il codice
  - L'interfaccia `List` definisce i metodi `add()`, `get()`, `size()`, `remove()`, etc.
  - La classe `ArrayList` è una classe che implementa l'interfaccia `List`
- Un princìpio di Ingegneria del software suggerisce: **programmare per le interfacce e non per le implementazioni**
  - Permette di ridurre la propagazione dei cambiamenti quando ci sarà necessità di adattare il codice
  - Permette di implementare più facilmente i test
- Le librerie Java forniscono tanti tipi (Collection) che sono contenitori di dati: liste, code, alberi, etc. Ognuno di questi è parametrizzato con il tipo di dato che deve contenere

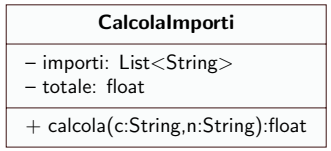
Un Esempio Pratico

- Riusciamo a sviluppare un componente software che risulti: riusabile, modificabile e corretto?
- Consideriamo un componente estremamente piccolo (potrebbe far parte di un sistema più grande)
- Descrizione dei **requisiti**
  - Dati vari file contenenti valori numerici, con un valore per ciascuna riga del file
    1. Leggere da ciascun file la lista di valori
    2. Tenere solo i valori non duplicati
    3. Calcolare la somma dei numeri letti dal file (non duplicati)
    4. Calcolare il massimo fra i numeri letti

```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
    }
}
```



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

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

Progressi

- Passi implementati
  - lettura da file
  - calcolo del totale
- Da fare
  - controlli su valori unici
  - estrazione del massimo

```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
    }
}
```


Progressi

- Passi implementati
  - lettura da file
  - calcolo del totale
  - estrazione del massimo
- Da fare
  - controlli su valori unici


```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;
    }
}
```


Progressi

- Passi implementati
  - lettura da file
  - calcolo del totale
  - estrazione del massimo
  - controlli su valori unici
- Il codice è conforme alla programmazione OO?
- E se il codice prodotto fosse invece …

```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;
    }
}
```


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

Spaghetti Code

- Metodi lunghi, senza parametri, e che usano variabili globali
- Flusso di esecuzione determinato dall'implementazione interna all'oggetto, non dai chiamanti
- Interazioni minime fra oggetti
- Nomi classi e metodi indicano la programmazione procedurale
- Ereditarietà e polimorfismo non usati, riuso impossibile
- Gli oggetti non mantengono lo stato fra le invocazioni
- Cause: inesperienza con OOP, nessuna progettazione


```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;                                                    leggiFile()
            while ((riga = f.readLine()) != null)
                if (!importi.contains(riga)) importi.add(riga);             inserisci()
            f.close();
        } catch (IOException e) { System.err.println("Errore di I/O: " + e.getMessage()); }
        // calcolo del totale                                               calcolaSomma()
        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                                           calcolaMassimo()
        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;                                                      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()
    }
}
```



```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;
        }
    }
```

Pagamenti

- importiLetti: List<String>
- valori: List<Float>
- totale: float
- massimo: float

+ leggiFile(c: String, n: String)
+ inserisci(valore: String)
+ converti()
+ calcolaSomma()
+ calcolaMassimo()
+ svuota()
+ getMassimo() : float
+ getSomma() : float

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


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`

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

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