Laboratorio di Programmazione

Lezione 14 – OOP

Marco Anisetti (teoria)
Dipartimento di Informatica
marco.anisetti@unimi.it

Matteo Luperto (lab. turno A)
Dipartimento di Informatica
matteo.luperto@unimi.it

Nicola Bena (lab. turno B)
Dipartimento di Informatica
nicola.bena@unimi.it

Object-Oriented Programming

- Supera la programmazione imperativa definendo il programma in termini di *oggetti* che interagiscono tra loro
- La struttura di un oggetto è definita da una classe in termini di
  - Attributi: variabili che modellano lo stato dell'oggetto
  - Metodi: funzioni che compiono delle azioni sulla base dello stato dell'oggetto
- Evoluzione del concetto di `struct` e di funzioni che operano su una variabile di tipo `struct`
  - Oggetto = istanza di una classe = variabile di tipo *`classe`*

Object-Oriented Programming-*ish* in C

```c
struct LinkedList {
  Node *head; int size;
};

void addElement(struct LinkedList *list, int value)
{
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = value;
  newNode->next = NULL;
  if (list->head == NULL) {
    list->head = newNode;
  } else {
    Node *current = list->head;
    while (current->next != NULL) {
      current = current->next;
    }
    current->next = newNode;
  }
  list->size++;
}

int main(){
  struct LinkedList list;
}
```

Object-Oriented Programming-*ish* in C

```c
struct LinkedList {
  Node *head; int size;
};
```

Definizione `struct LinkedList`

```c
void addElement(struct LinkedList *list, int value)
{
  Node *newNode = (Node *)malloc(sizeof(Node));
  newNode->data = value;
  newNode->next = NULL;
  if (list->head == NULL) {
    list->head = newNode;
  } else {
    Node *current = list->head;
    while (current->next != NULL) {
      current = current->next;
    }
    current->next = newNode;
  }
  list->size++;
}
```

Definizione di una funzione che opera su una variabile di tipo `struct LinkedList*`

```c
int main(){
  struct LinkedList list;
}
```

Dichiarazione di una variabile di tipo `struct LinkedList`

Object-Oriented Programming in Java: Classe

- La definizione della struct viene sostituita dalla definizione della classe
  - Attributi corrispondono ai campi della `struct`
  - La visibilità definisce chi può accedere direttamenente (o no) agli attributi
  - Principio di incapsulamento
    - Definendo gli attributi come `private` nascondiamo a chi usa la classe come la classe è stata implementata
    - Il programmatore deve conoscere solo il prototipo dei metodi della classe
  - Metodi sono funzioni che agiscono sull'istanza della classe, eventualmente modificandone lo stato interno modificando gli attributi
- Oggetto = istanza della classe `MyClass` = variabile di tipo `MyClass`
  - Dopo aver istanziato la classe, posso usarne i relativi metodi
    - Sintassi: `nomeVariabile.nomeMetodo(parametri)`

Object-Oriented Programming in Java: Classe

- **Attributi** `private`
- **Metodi** `public` **(o** `protected`**)**

```java
public class Person {
  public String firstName;
  public String lastName;
  public String SSN;

  public Person(String name1, String name2, String ssn){
    firstName = name1;
    lastName = name2;
    SSN = ssn;
  }
}
```

```java
public class Person {
  private String firstName;
  private String lastName;
  private String SSN;
  public Person(String name1, String name2, String ssn){
    firstName = name1;
    lastName = name2;
    SSN = ssn;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String name) {
    firstName = name;
  }
}
```

Object-Oriented Programming in Java: Costruttore

- Metodo che crea un'istanza della classe
  - Definisce come inizializzare i suoi campi
  - Il nome del costruttore corrisponde al nome della classe
  - La chiamata al costruttore alloca spazio sull'heap per l'oggetto e ne restituisce il puntatore (trasparente al programmatore)
  - → Chiamare il costruttore significa creare un'istanza di quella classe
- Aggiunto di default se non specificato
  - Inizializza un oggetto "nullo" (esempio: una stringa String vuota)

```
MyClass variabile = new MyClass(parametri)
```

Object-Oriented Programming in Java: Esempio

```c
struct Person {
  char first_name[100];
  char last_name[100];
  char ssn[15];
};


int main(){
  struct Person * persona =
       (struct Person*) malloc(
       sizeof(struct Person));

  printf(persona->first_name);
}
```

```java
public class Person {
  private String firstName;
  private String lastName;
  private String SSN;
  public String getFirstName() {
    return firstName;
  }
}

public class Main {
  public static void main(String[] args){
    Person persona = new Person();

    System.out.println(persona.getFirstName());
  }
}
```

Object-Oriented Programming in Java: Esempio

Definizione campi

```c
struct Person {
  char first_name[100];
  char last_name[100];
  char ssn[15];
};
```

Definizione attributi e metodi

```java
public class Person {
  private String firstName;
  private String lastName;
  private String SSN;
  public String getFirstName() {
    return firstName;
  }
}
```

Allocazione esplicita (senza inizializzazione campi)

```c
int main(){
  struct Person * persona =
      (struct Person*) malloc(
      sizeof(struct Person));

  printf(persona->first_name);
}
```

Accesso diretto a un campo

```java
public class Main {
  public static void main(String[] args){
    Person persona = new Person();
    System.out.println(persona.getFirstName());
  }
}
```

Istanziazione (costruttore implicito)

Accesso a un campo solo tramite metodo

Object-Oriented Programming in Java: Metodi

- Metodo: funzione che opera su un oggetto, ovvero legge e scrive i valori dei suoi attributi
  - Il metodo viene chiamato su un oggetto, quindi nel corpo del metodo abbiamo accesso ai suoi attributi
  - Keyword `this` usata per riferirsi all'istanza dell'oggetto su cui il metodo viene invocato, utile per disambiguare in caso di un parametro in ingresso con lo stesso nome di un attributo
- Definizione metodi *getter* e *setter* se è necessario accedere/modificare gli attributi dopo l'istanziazione

Object-Oriented Programming in Java: This

```
public class Person {
  private String firstName;
  private String lastName;
  private String SSN;

  public Person(String firstName, String lastName,
         String SSN){
    this.firstName = firstName;
    this.lastName = lastName;
    this.SSN = SSN;
  }

  public String getFirstName() {
    return firstName;
  }
  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }
}
```

I nomi dei parametri in ingresso sono uguali ai nomi degli attributi: **`this.<nome>`** si riferisce all'attributo `<nome>` dell'oggetto creato

Metodo `get<NomeAttributo>`: restituisce l'attributo; Permette di controllare/gestire l'accesso al dato, filtrando accesso al «vero» valore e implementazione della classe

Metodo `set<NomeAttributo>`: dà un valore all'attributo Permette di validare il valore ricevuto in ingresso, proteggendo i dati

Object-Oriented Programming in Java: This

I nomi dei parametri non causano ambiguità, quindi `this` non è strettamente necessario

Suggerimento

Per evitare confusione, usatelo sempre

```
public class Person {
  private String firstName;
  private String lastName;
  private String SSN;

  public Person(String name1, String name2, String ssn){
    firstName = name1;
    lastName = name2;
    SSN = ssn;
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String name) {
    firstName = name;
  }
}
```

Oggetti e Riferimenti

- Gli oggetti sono sempre passati ai metodi per riferimento
  - Formalmente, è un passaggio per valore del puntatore all'oggetto
- Lo stesso vale per il valore restituito da un metodo
  - Se un metodo restituisce un oggetto attributo dell'oggetto corrente, le modifiche all'oggetto restituito si riflettono nell'attributo

Oggetti e Riferimenti

```
import java.util.ArrayList;

public class Group {

  private ArrayList<Person> people;

  private String name;
  public Group(String name){
    this.people = new ArrayList<Person>();
    this.name = name;
  }

  public Group(String name,
         ArrayList<Person> people){
    this.people = people;
    this.name = name;
  }

  public ArrayList<Person> getPeople(){
    return this.people;
  }
```

```
  public static void main(String[]args){
    Person p1 = new Person("Mario", "Rossi", "123");
    Person p2 = new Person("Mario", "Bianchi", "456");

    ArrayList<Person> people = new ArrayList<Person>();
    people.add(p1);
    people.add(p1);

    Group group = new Group("Marii", people);

    people = group.getPeople();
    people.clear();

    System.out.println(group.people.size());
   }
  }
```

Oggetti e Riferimenti

```java
import java.util.ArrayList;

public class Group {

  private ArrayList<Person> people;

  private String name;
  public Group(String name){
    this.people = new ArrayList<Person>();
    this.name = name;
  }

  public Group(String name,
         ArrayList<Person> people){
    this.people = people;
    this.name = name;
  }

  public ArrayList<Person> getPeople(){
    return this.people;
  }
```

```java
public static void main(String[]args){
  Person p1 = new Person("Mario", "Rossi", "123");
  Person p2 = new Person("Mario", "Bianchi", "456");

  ArrayList<Person> people = new ArrayList<Person>();
  people.add(p1);
  people.add(p1);

  Group group = new Group("Marii", people);

  people = group.getPeople();
  people.clear();

  System.out.println(group.people.size());
 }
}
```

Restituisce `people`, cioè un puntatore a `ArrayList` (tipo `ArrayList*` in C)

Oggetti e Riferimenti

```java
import java.util.ArrayList;

public class Group {

  private ArrayList<Person> people;
  private String name;

  public Group(String name){
    this.people = new ArrayList<Person>();
    this.name = name;
  }

  public Group(String name,
         ArrayList<Person> people){
    this.people = people;
    this.name = name;
  }

  public ArrayList<Person> getPeople(){
    return this.people;
  }
```

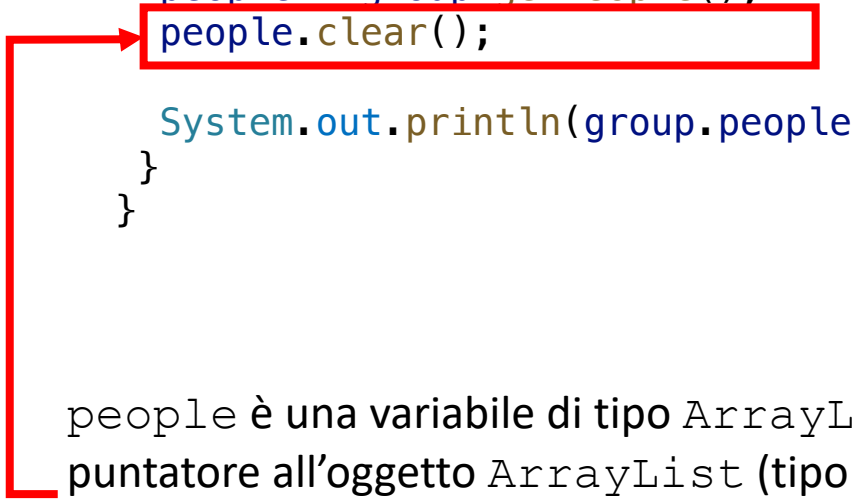
```java
  public static void main(String[]args){
    Person p1 = new Person("Mario", "Rossi", "123");
    Person p2 = new Person("Mario", "Bianchi", "456");

    ArrayList<Person> people = new ArrayList<Person>();
    people.add(p1);
    people.add(p1);

    Group group = new Group("Marii", people);

    people = group.getPeople();
    people.clear();

    System.out.println(group.people.size());
  }
}
```

`people` è una variabile di tipo `ArrayList`, cioè un puntatore all'oggetto `ArrayList` (tipo `ArrayList*` in C)

Oggetti e Riferimenti

```java
import java.util.ArrayList;

public class Group {

  private ArrayList<Person> people;
  private String name;

  public Group(String name){
    this.people = new ArrayList<Person>();
    this.name = name;
  }

  public Group(String name,
         ArrayList<Person> people){
    this.people = people;
    this.name = name;
  }

  public ArrayList<Person> getPeople(){
    return this.people;
  }
```

```java
  public static void main(String[]args){
    Person p1 = new Person("Mario", "Rossi", "123");
    Person p2 = new Person("Mario", "Bianchi", "456");

    ArrayList<Person> people = new ArrayList<Person>();
    people.add(p1);
    people.add(p1);

    Group group = new Group("Marii", people);

    people = group.getPeople();
    people.clear();

    System.out.println(group.people.size());
  }
}
```



`people` è una variabile di tipo `ArrayList`, cioè un puntatore all'oggetto `ArrayList` (tipo `ArrayList*` in C)
Le modifiche sono applicate sullo stesso oggetto, quindi anche su `group.people`

Modificatore `static`

- Attributo: c'è un unica istanza dell'attributo, comune e condivisa tra tutte le istanze della classe
- Metodo: metodo di classe e non di istanza → può essere chiamato senza istanziare la classe
  - Sintassi: `MyClass.myMethod(parametri)`
  - → motivo per cui definivamo metodi `static` negli esercizi fatti fin'ora
  - Non è possibile usare `this` nel corpo di un metodo `static`
  - Simili alle funzioni in C – vivono senza un oggetto

Modificatore `static`

```
public class Obj {

  private static int contatore = 0;

  public Obj(){
    this.contatore++;
  }

  public int getContatore(){
    return this.contatore;
  }

  public static void main(String[] args){
    Obj o1 = new Obj();
    Obj o2 = new Obj();
    System.out.printf("Contatore: %d\n",
       o2.getContatore());
  }
}
```

Modificatore `static`

```
public class Obj {

  private static int contatore = 0;

  public Obj(){
    this.contatore++;
  }

  public int getContatore(){
    return this.contatore;
  }

  public static void main(String[] args){
    Obj o1 = new Obj();
    Obj o2 = new Obj();
    System.out.printf("Contatore: %d\n",
       o2.getContatore());
  }
}
```

Modificatore `final`

- Attributo: il valore dell'attributo non può essere modificato (se è un oggetto, non posso cambiare il puntatore)
  - Inizializzato nella dichiarazione oppure nel costruttre
- Metodo: il metodo non può essere sovrascritto da una classe figlia
- → Attributi `static final` corrispondono a costanti

```
private static final int LENGTH = 10;
```

Cartelle e Import

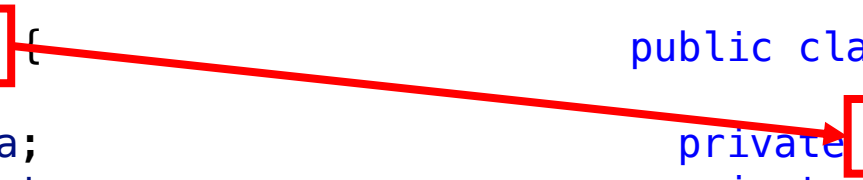
- Una classe può essere usata all'interno di un'altra classe se si trovano nella stessa cartella ("package di default"), e se hanno visibilità corretta
- Non sono necessari `import`

File `List/Node.java`

```
public class Node {

  private int data;
  private Node next;

  public Node(int data){
    this.data = data;
    this.next = null;
}
```

File `List/LinkedList.java`

```
public class LinkedList {

  private Node head;
  private int size;

  public LinkedList() {
    this.head = null;
    this.size = 0;
}
```



Object-Oriented Programming in Java

```
public class Node {

  private int data;
  private Node next;

  public Node(int data){
    this.data = data;
    this.next = null;
  }

  public boolean hasNext(){
    return this.next != null;
  }

  public Node getNext(){
    return this.next;
  }

  public void setNext(Node next){
    this.next = next;
  }
}
```

Attributi

Costruttore

Metodo che opera sulla classe → definito all'interno della classe (maggiore coesione)

Getter e setter: la modifica dello stato di un oggetto avviene solo tramite metodi, i suoi dettagli implementativi sono nascosti (attributi `private`)

Object-Oriented Programming in Java

```
public class LinkedList {

  private Node head;
  private int size;

  public LinkedList() {
    this.head = null;
    this.size = 0;
  }

  public void addElement(int value){
    Node newNode = new Node(value);
    if (this.head == null){
      this.head = newNode;
    } else {
      Node current = this.head;
      while (current.hasNext()){
        current = current.getNext();
      }
      current.setNext(newNode);
   }
   this.size++;
  }
}
```

Metodo che opera sulla classe → definito all'interno della classe (maggiore coesione)

Convenzioni: `toString`

- Metodo che restituisce una `String` che rappresenta in forma testuale la classe
- Il formato è libero della stringa è libero, la prassi prevede di includere (almeno) il valore degli attributi più significativi
- Utile anche come strategia di debugging
- Aggiunto automaticamente se non definito da noi

Convenzioni: `toString`

- Metodo che restituisce una `String` che rappresenta in forma testuale la classe
- Il formato è libero della stringa è libero, la prassi prevede di includere (almeno) il valore degli attributi più significativi
- Utile anche come strategia di debugging
- Aggiunto automaticamente se non definito da noi

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }

  public static void main(String[] args){
    Fraction fraction = new Fraction(5, 4);
    System.out.println(fraction.toString());
  }
}
```

Convenzioni: `toString`

- Metodo che restituisce una `String` che rappresenta in forma testuale la classe
- Il formato è libero della stringa è libero, la prassi prevede di includere (almeno) il valore degli attributi più significativi
- Utile anche come strategia di debugging
- Aggiunto automaticamente se non definito da noi

```java
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }

  public static void main(String[] args){
    Fraction fraction = new Fraction(5, 4);
    System.out.println(fraction.toString());
  }
}
```

```
❯ java Fraction
Fraction@75a1cd57
```

Convenzioni: `toString`

- Metodo che restituisce una `String` che rappresenta in forma testuale la classe
- Il formato è libero della stringa è libero, la prassi prevede di includere (almeno) il valore degli attributi più significativi
- Utile anche come strategia di debugging
- Aggiunto automaticamente se non definito da noi

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }

  public String toString(){
    return String.format("%d/%d", this.num,
        this.den);
  }

  public static void main(String[] args){
    Fraction fraction = new Fraction(5, 4);
    System.out.println(fraction.toString());
  }
}
```

```
> java Fraction
5/4
```

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a ==)

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a ==)

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }
  public String toString(){
    return String.format("%d/%d", this.num, this.den);
  }

  public static void main(String[] args){
    Fraction f1 = new Fraction(5, 4);
    Fraction f2 = new Fraction(5, 4);
    Fraction f3 = f1;
    System.out.printf("f1(%s) equals f2(%s)? %b\n",
        f1.toString(), f2.toString(), f1.equals(f2));
    System.out.printf("f1(%s) equals f3(%s)? %b\n",
        f1.toString(), f3.toString(), f1.equals(f3));
  }
}
```

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a ==)

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }
  public String toString(){
    return String.format("%d/%d", this.num, this.den);
  }

  public static void main(String[] args){
    Fraction f1 = new Fraction(5, 4);
    Fraction f2 = new Fraction(5, 4);
    Fraction f3 = f1;
    System.out.printf("f1(%s) equals f2(%s)? %b\n",
        f1.toString(), f2.toString(), f1.equals(f2));
    System.out.printf("f1(%s) equals f3(%s)? %b\n",
        f1.toString(), f3.toString(), f1.equals(f3));
  }
}
```

```
> java Fraction
f1(5/4) equals f2(5/4)? false
f1(5/4) equals f3(5/4)? true
```

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a ==)

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }
  public boolean equals(Fraction other){
    return this.num/this.den == other.num/other.den;
  }
  public String toString(){
    return String.format("%d/%d", this.num, this.den);
  }

  public static void main(String[] args){
    Fraction f1 = new Fraction(5, 4);
    Fraction f2 = new Fraction(10, 8);
    Fraction f3 = new Fraction(5, 4);
    System.out.printf("f1(%s) equals f2(%s)? %b\n",
         f1.toString(), f2.toString(), f1.equals(f2));
    System.out.printf("f1(%s) equals f3(%s)? %b\n",
         f1.toString(), f3.toString(), f1.equals(f3));
  }
}
```

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a ==)

```
public class Fraction {
  private int num;
  private int den;
  public Fraction(int num, int den){
    this.num = num;
    this.den = den;
  }
  public boolean equals(Fraction other){
    return this.num/this.den == other.num/other.den;
  }
  public String toString(){
    return String.format("%d/%d", this.num, this.den);
  }

  public static void main(String[] args){
    Fraction f1 = new Fraction(5, 4);
    Fraction f2 = new Fraction(10, 8);
    Fraction f3 = new Fraction(5, 4);
    System.out.printf("f1(%s) equals f2(%s)? %b\n",
        f1.toString(), f2.toString(), f1.equals(f2));
    System.out.printf("f1(%s) equals f3(%s)? %b\n",
        f1.toString(), f3.toString(), f1.equals(f3));
  }
}
```

```
> java Fraction
f1(5/4) equals f2(10/8)? true
f1(5/4) equals f3(5/4)? true
```

Convenzioni: `equals`

- Metodo che restituisce un boolean che indica se due oggetti sono *uguali* (`true`) o meno (`false`)
- L'implementazione del metodo determina cosa si intende per uguaglianza (ad es., confronto un sottinsieme degli attributi)
- Prototipo: `boolean equals(MyClass other)`, dove `other` è un oggetto che istanzia la classe corrente (oppure di tipo `Object`)
- Aggiunto automaticamente se non definito da noi, ed esegue un confronto sui puntatori (equivalente a `==`)

Se l'implementazione di `equals` (o di un altro metodo "noto") non è ovvia, ovvero non confronta tutti gli attributi usando il rispettivo metodo `equals` per ciascuno, è bene segnalarlo nella documentazione, oppure dare un nome diverso al metodo, ad esempio `equalsOnValue`

Quali Metodi? Quali Attributi?

- Definire metodi ed attributi di una classe è uno step particolarmente importante nel design di un progetto Java
- Di solito le specifiche sono diretta conseguenza dell'uso che se ne vuole fare, aka non c'è una unica soluzione giusta
  (ma la vostra soluzione deve essere giusta)
- I metodi ed attributi da definire sono *tutti e soli* quelli che servono per il progetto specifico
- Come visibilità/accessibilità vale sempre *il principio del privilegio minimo* (Principle of Least Privilege), ovvero che ogni modulo abbia visibilità delle sole risorse/informazioni immediatamente necessarie al suo funzionamento

Esempio: la classe Libro

Quali attributi?

- Titolo
- Autore
- ISBN

Esempio: la classe Libro

Quali attributi?

- Titolo
- Autore
- ISBN
- Pagine
- Edizione
- Tipo di copertina

Esempio: la classe Libro

Quali attributi?

- Titolo
- Autore
- ISBN
- Pagine
- Edizione
- Tipo di copertina
- Lingua
- Peso
- Altezza
- Larghezza
- Spessore
- Colore
- Stato di conservazione
- Prezzo
- Rarità
- Ancora in commercio?
- Argomento
- Personaggi
- Rating (adulti, ragazzi, …)
- Valutazione
- Sconto
- Nuovo / usato
- Recensioni
- Errori grammaticali

… e potremmo andare avanti …

Esempio: la classe Libro *per una* Biblioteca

Quali attributi?

- Titolo
- Autore
- (ISBN)
- Status (disponibile o meno)

```
public class Book {
        private String title;
        private String author;
        private boolean isBorrowed;

public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isBorrowed = false;
}
```

```
public void borrowBook() {
        if (isBorrowed) {
                System.out.println(title + " non disponibile!");
        } else {
                isBorrowed = true;
                System.out.println("Prestito: " + title);
        }
}

public void returnBook() {
        if (!isBorrowed) {
                System.out.println("Non in prestito.");
        } else {
        isBorrowed = false;
        System.out.println("Grazie per aver restituito: " + title);
}
```

Esempio: la classe Libro *per una* Biblioteca

Quali attributi?

- Titolo
- Autore
- (ISBN)
- Status (disponibile o meno)

```java
public class Book {
        private String title;
        private String author;
        private boolean isBorrowed;

public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isBorrowed = false;
}
```

```java
public void borrowBook() {
        if (isBorrowed) {
                System.out.println(title + " non disponibile!");
        } else {
                isBorrowed = true;
                System.out.println("Prestito: " + title);
        }
}

public void returnBook() {
        if (!isBorrowed) {
                System.out.println("Non in prestito.");
        } else {
        isBorrowed = false;
        System.out.println("Grazie per aver restituito: " + title);
}
```

getter e setter dello status del libro

Un solo costruttore, no parametri di default

Esempio: la classe Libro *per una* Biblioteca

Quali attributi?

- Titolo
- Autore
- (ISBN)
- Status (disponibile o meno)

```
public class Book {
        private String title;
        private String author;
        private boolean isBorrowed;

public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isBorrowed = false;
}
```

Cosa (si potrebbe) aggiungere?

- Timestamp della data di prestito (e lo storico)
- Chi ha preso in prestito
- Lo status del libro (al momento della prestito)
- Copie duplicate
- Prenotazioni di prestiti
- …

Tutte informazioni specifiche per il caso di uso - biblioteca

Metodi e attributi

- Astrazione: ignorare i dettagli irrilevanti e concentrarsi solo su quelli specifici per il contesto applicativo
- Rilevanza: se il programma **non** usa, **non** mostra, o **non** ha bisogno di un attributo per prendere una decisione, questo attributo **non** va rappresentato nella classe
  esempio: il numero di errori ortografici nel libro per la biblioteca
- Incapsulamento:
  `isBorrowed` è lo stato del libro, ma il metodo `borrowBook()` è quello che setta/cambia lo stato, proteggendo i dati

Esempio 2: media voti

Classe che gestisce il vostro *curriculum* universitario:

- Quanti esami fatti
- Quanti crediti vi rimangono
- Media voto (in 30esimi)
- Valutazione finale (in 110imi)

Metodo o attributo?

```
private float media;
private float votoFinale;
```

oppure

```
public float getMedia();
public float getVotoFinale();
```

(per brevità vediamo una versione semplificata, tutti gli esami valgono lo stesso numero di crediti, non devo tenere traccia degli esami fatti ma solo del voto ricevuto)

Esempio 2: media voti

```
public class StudentGrades {
  private String studentName;
  private List<Integer> grades;

public StudentGrades(String studentName) {
   this.studentName = studentName;
   this.grades = new ArrayList<>();
}

public void addGrade(int grade) {
  if (grade < 18 || grade > 30) {
        System.out.println("Grade must be between 18 and 30");
        grades.add(grade);
  }

}
```

Esempio 2: media voti

Opzione 1: calcolo ogni volta il valore medio e il voto previsto

```
public double getAverage() {
  if (grades.isEmpty()) return 0;
  int sum = 0;
  for (Integer g : grades) {
    sum += g;
  }
  return (double) sum / grades.size();
}
public double getExpectedGraduationMark() {
  double average = getAverage();
  return (average / 30.0) * 110.0;
}
```

Esempio 2: media voti

Opzione 2: salvo in un attributo media e/o il voto previsto, e lo aggiorno di volta in volta

```
public double getAverage() {
  return average;
}
public double getGraduationMark() {
return graduationMark;
}
```

```
private double average;
private double graduationMark;

public void addGrade(int grade) {
  if (grade < 18 || grade > 30) {
        System.out.println("Grade must be between
18 and 30");
  }
  grades.add(grade);
  recalculate();
}

private void recalculate() {
  if (grades.isEmpty()) {
        average = 0;
        graduationMark = 0;
        return;
  }

  int sum = 0;
  for (Integer g : grades) {
        sum += g;
  }

  average = (double) sum / grades.size();
  graduationMark = (average / 30.0) * 110.0;
}
```

Esempio 2: media voti

- Rappresento esplicitamente la media
- Devo aggiornarla in maniera consistente ad ogni update
  - Tenendo traccia di tutti i casi, cosa succeed se faccio un orale e il mio voto cambia?
- Metodo privato, non serve esporlo
- Accesso al dato è immediato

```
private double average;
private double graduationMark;

public void addGrade(int grade) {
  if (grade < 18 || grade > 30) {
        System.out.println("Grade must be between
18 and 30");
  }
  grades.add(grade);
  recalculate();
}

private void recalculate() {
  if (grades.isEmpty()) {
        average = 0;
        graduationMark = 0;
       return;
  }

  int sum = 0;
  for (Integer g : grades) {
        sum += g;
  }

  average = (double) sum / grades.size();
  graduationMark = (average / 30.0) * 110.0;
}
```

```java
public double getAverage() {
  if (grades.isEmpty()) return 0;
  int sum = 0;
  for (Integer g : grades) {
       sum += g;
  }
  return (double) sum / grades.size();
 }

public double getExpectedGraduationMark()
{
      double average = getAverage();
      return (average / 30.0) * 110.0;
}
```

```java
private double average;
private double graduationMark;

public double getAverage() {
return average;
}

public double getGraduationMark() {
return graduationMark;
}
```

Quale soluzione migliore? *Dipende*

- Se i voti cambiano spesso, e calcolo raramente la media: Soluzione 1
- Se i voti cambiano poco, e devo accedere alla media spesso: Soluzione 2

Java in VS Code

- Layout: una cartella per esercizio
- Per evitare errori nei suggerimenti, in VS Code aprite solo la cartella dell'esercizio corrente
  - Oppure ignorare i suggerimenti dell'IntelliSense e affidatevi solo al compilatore
  - Usate i package solo se siete sicuri