

Lezione 10 - Classi e orientamento agli oggetti (Parte 2)

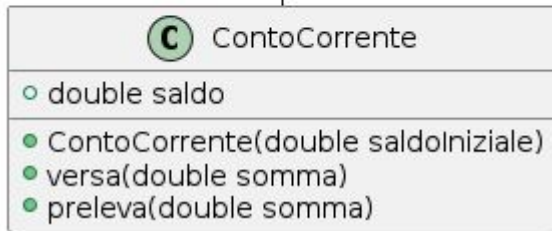
Sylvio Barbon Junior
sylvio.barbonjunior@units.it

Sommario:

Lezione scorsa;

- 1) L'esecuzione di un programma a oggetti
- 2) Incapsulamento
 - a) Esempi

1) Esempi



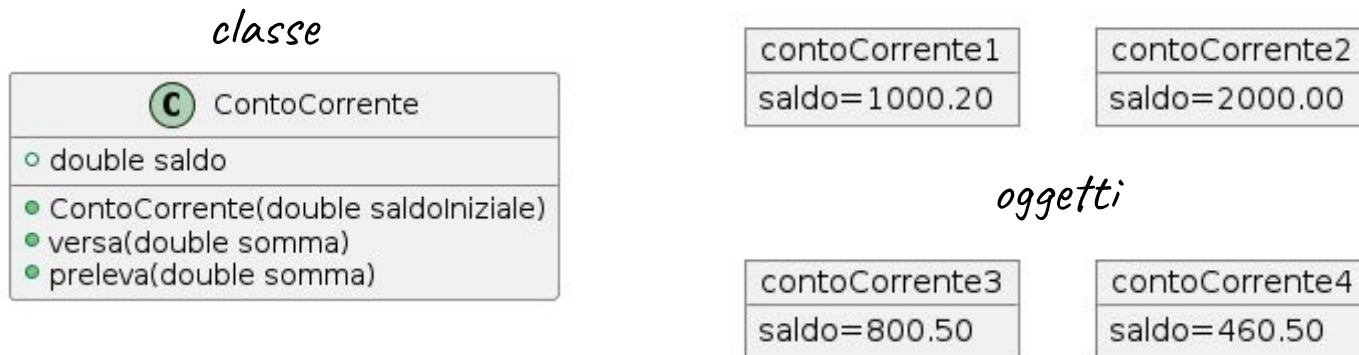
```
1 public class UsaConto {
2     public static void main ( String [] args ) {
3         ContoCorrente cc = new ContoCorrente (1000);
4
5         cc.versa(700);
6
7         if (cc.saldo >200)
8             cc.preleva(200);
9         if (cc.saldo >900)
10            cc.preleva(900);
11        System.out.println("Saldo finale:"+cc.saldo);
12    }
13 }
```

```
1 public class ContoCorrente {
2
3     public double saldo ;
4
5     public ContoCorrente ( double saldoIniziale ) {
6         saldo = saldoIniziale ;
7     }
8
9     public void versa ( double somma ) {
10        saldo += somma ;
11    }
12
13    public void preleva ( double somma ) {
14        saldo -= somma ;
15    }
16 }
```

```
barbon@barbon-XPS13-9333: ~/Downloads/FI/uml$
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$
Saldo finale:600.0
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$
```

1) L'esecuzione di un programma a oggetti:

- In un **programma basato su oggetti**, lo stato non è più unico e globale come nella programmazione imperativa. Invece, è composto da tutti gli stati interni di tutti gli oggetti attivi.
- Questo consente una maggiore **modularità** e la possibilità **di isolare e gestire** singolarmente **gli stati degli oggetti**, migliorando la manutenibilità e la flessibilità del codice.



1) L'esecuzione di un programma a oggetti:

La programmazione orientata agli oggetti semplifica **la realizzazione di programmi complessi**:

- a) Si identificano le **varie entità** da rappresentare tramite classi e oggetti;
- b) Si specificano le variabili e i metodi di ogni classe accessibili dalle altre classi (ossia **l'interfaccia pubblica della classe**);
- c) Si implementano le varie classi separatamente, **concentrandosi su una per volta**;
- d) Le varie classi possono essere implementate da **persone diverse indipendentemente**;
- e) Più **facile fare manutenzione e aggiornamenti** a parti del programma!

1) L'esecuzione di un programma a oggetti:

- **L'incapsulamento** consiste nella protezione dell'implementazione;
- **L'ereditarietà** permette essenzialmente di definire delle classi a partire da altre già definite;
- **Il polimorfismo** permette di scrivere **un client** che può servirsi di oggetti di classi diverse;

2) Incapsulamento

Un modo differente di pensare le cose quando si programma:

- **ogni oggetto è indipendente** e comunica con gli altri attraverso lo scambio di **messaggi**;
- **vantaggio**: costruzione di un oggetto e suo riutilizzo.

grande flessibilità

Nella programmazione orientata agli oggetti, solitamente si mette in **stretta relazione** un pezzo di informazione con il comportamento specifico, **che agisce** su tale informazione

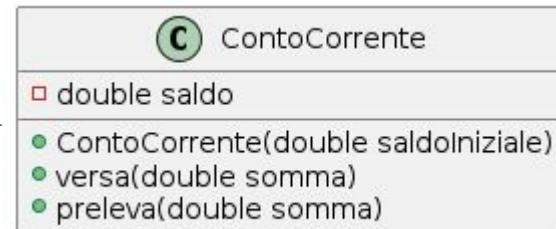
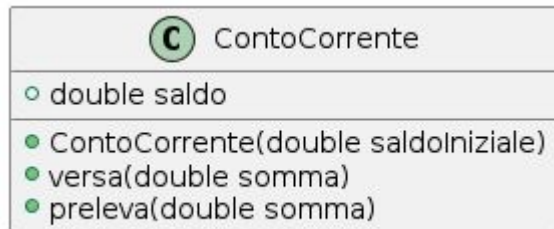
metodo

2) Incapsulamento

L'**incapsulamento** è proprio legato al concetto di “**impacchettare**” in un oggetto i dati e le azioni che sono riconducibili ad un singolo componente.

Il saldo è pubblico (**public**)

se vogliamo evitare che sia modificabile dall'esterno della classe lo dobbiamo trasformare in privato (**private**)



2) Incapsulamento

Ora siamo sicuri che le modifiche al saldo avverranno solo **tramite i metodi**.
Come farà il main a stampare il saldo?


```
1 public class UsaConto {
2     public static void main ( String [] args ) {
3         ContoCorrente cc = new ContoCorrente (1000);
4
5         cc.versa(700);
6
7         if (cc.saldo < 200)
8             cc.preleva(200);
9         if (cc.saldo < 900)
10            cc.preleva(900);
11         System.out.println("Saldo finale:"+cc.saldo);
12     }
13 }
```

2) Incapsulamento tramite i **metodi**

```
1 public class UsaConto {
2     public static void main ( String [] args ) {
3         ContoCorrente cc = new ContoCorrente (1000);
4
5         cc.versa(700);
6
7         if (cc.ottieniSaldo() >200)
8             cc.preleva(200);
9         if (cc.ottieniSaldo() >900)
10            cc.preleva(900);
11        System.out.println("Saldo finale:"+cc.ottieniSaldo());
12    }
13 }
```

2) Incapsulamento

```
1 public class ContoCorrente {
2
3     private double saldo ;
4
5     public ContoCorrente ( double saldoIniziale ) {
6         saldo = saldoIniziale ;
7     }
8
9     public void versa ( double somma ) {
10        saldo += somma ;
11    }
12
13    public void preleva ( double somma ) {
14        saldo -= somma ;
15    }
16
17    public double ottieniSaldo () {
18        return this.saldo ;
19    }
20 }
```

 ContoCorrente
□ saldo:double
● ContoCorrente(double saldoIniziale)
● versa(double somma)
● preleva(double somma)
● ottieniSaldo () :double

parola riservata this

2) Incapsulamento Esempi

Ⓒ Car
❑ name:String
❑ topSpeed:double
❑ currentSpeed:double
● Car(String)
● Car(String, double)
● getName():String
● setName(String)
● speedUP();
● slowDown();
● getCurrentSpeed():double

```
1 public class Car{
2     private String name;
3     private double topSpeed;
4     private double currentSpeed;
5
6     public Car(String name){
7         this.name = name;
8         this.currentSpeed = 0;
9         this.topSpeed = 200.00;
10    }
11
12    public Car(String name, double topSpeed){
13        this.name = name;
14        this.topSpeed = topSpeed;
15    }
16
17    public String getName(){
18        return(this.name);
19    }
20
21    public void setName(String name){
22        this.name = name;
23    }
24
25    public void speedUp(){
26        this.currentSpeed++;
27    }
28
29    public void slowDown(){
30        this.currentSpeed--;
31    }
32
33    public double getCurrentSpeed(){
34        return this.currentSpeed;
35    }
36
37 }
```

← attributi (variabile) **privati**

← metodi costruttori (overload) - **public**

return con

parenthesis

← metodi "get" per recuperare i valori
incapsulati - **public**

← metodi "set" per fare modifica
nelle variabile - **public**

return senza

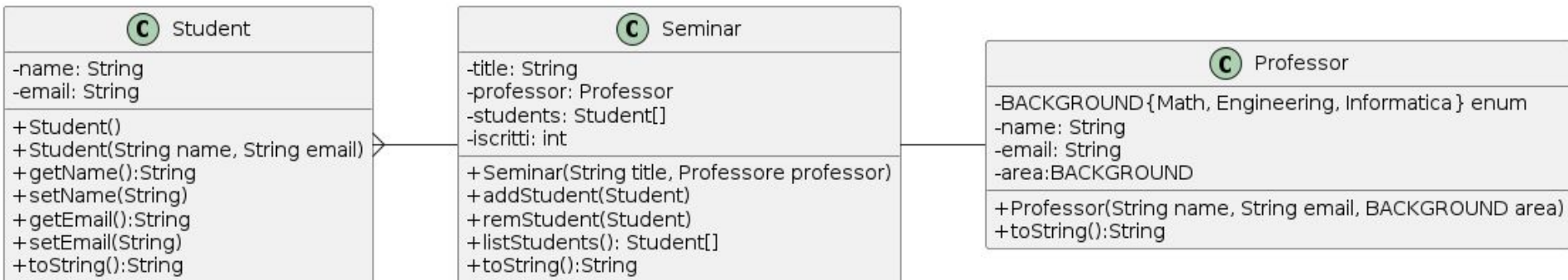
parenthesis

2) Incapsulamento

```
1 public class ExecuteCar{
2     public static void main(String args[]){
3         new ExecuteCar();
4     }
5
6     ExecuteCar(){
7         Car car500 = new Car("500");
8         Car carPunto = new Car("Punto");
9         Car carFerrari = new Car("Ferrari", 340.00);
10
11         Car cars[] = {car500, carPunto, carFerrari};
12
13         float lim = 120.0f;
14         for(int vel = 0; vel<lim; vel++){
15             for(int c = 0; c<cars.length; c++){
16                 cars[c].speedUp();
17             }
18         }
19
20         for(int c = 0; c<cars.length; c++){
21             System.out.println("Velocità "+cars[c].getName()+
22                 " = "+cars[c].getCurrentSpeed());
23         }
24     }
25 }
```

```
barbon@barbon-XPS13-9333: ~/Downloads/FI/uml
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$ java ExecuteCar
Velocità 500 = 120.0
Velocità Punto = 120.0
Velocità Ferrari = 120.0
(base) barbon@barbon-XPS13-9333:~/Downloads/FI/uml$
```

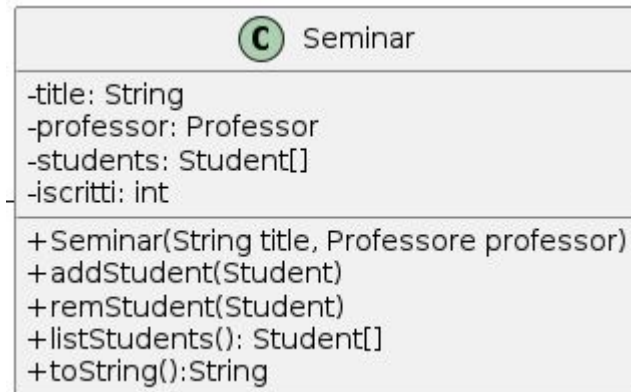
2) Incapsulamento Esempi



2) Incapsulamento

Esempi

```
1 public class Seminar{
2     private String title;
3     private Professor professor;
4     private Student[] students;
5     private int iscritti = 0;
6
7     public Seminar(String title, Professor professor, int n){
8         this.title = title;
9         this.professor = professor;
10        this.students = new Student[n];
11    }
12
13    public boolean addStudent(Student student){
14        if(iscritti < 20){
15            this.students[iscritti++] = student;
16            return(true);
17        }
18        return(false);
19    }
20 }
```



2) Incapsulamento

Esempi

```
21 public boolean remStudent(Student student){
22     Student[] auxStudents = new Student[students.length];
23     int aux = 0;
24     for(int i = 0; i<iscritti; i++){
25         if(!students[i].getName().equals(student.getName())){
26             auxStudents[aux++] = students[i];
27         }
28     }
29     students = auxStudents;
30     iscritti--;
31     if(aux==iscritti){
32         return(true);
33     }
34     return(false);
35 }
36
37 public Student[] listStudent(){
38     return(this.students);
39 }
40
41 public String toString(){
42     return(""+this.title+"\n by "+this.professor.toString());
43 }
44 }
```

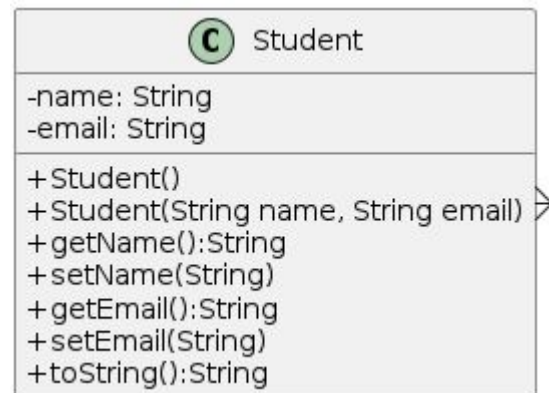
Seminar

-title: String
-professor: Professor
-students: Student[]
-iscritti: int

+Seminar(String title, Professore professor)
+addStudent(Student)
+remStudent(Student)
+listStudents(): Student[]
+toString():String

2) Incapsulamento


```
1 public class Student{
2     private String name;
3     private String email;
4     public Student(){
5     public Student(String name, String email){
6         this.name = name;
7         this.email = email;
8     }
9     public String getName(){
10        return(this.name);
11    }
12    public String getEmail(){
13        return(this.name);
14    }
15    public void setName(String name){
16        this.name = name;
17    }
18    public void setEmail(String email){
19        this.email = email;
20    }
21    public String toString(){
22        return(this.name+" ("+"this.email+"");
23    }
24 }
```



2) Incapsulamento

Esempi

```
1 public class Professor{
2     public enum BACKGROUND {MATH, ENGINEERING, INFORMATICA};
3
4     private String name;
5     private String email;
6     private BACKGROUND area;
7
8     public Professor(String name, String email, BACKGROUND area){
9         this.name = name;
10        this.email = email;
11        this.area = area;
12    }
13
14    public String toString(){
15        return(this.name+" ("+"+this.email+"") \n "+this.area);
16    }
17 }
```

 Professor

-BACKGROUND{Math, Engineering, Informatica} enum
-name: String
-email: String
-area:BACKGROUND

+Professor(String name, String email, BACKGROUND area)
+toString():String

2) Incapsulamento

Esempi

```
1 import java.util.Scanner;
2 public class MainSeminar{
3     Scanner sc = new Scanner(System.in);
4
5     public static void main(String[] args){
6         new MainSeminar();
7     }
```

```

9 public MainSeminar(){
10     Professor professor = new Professor("Sylvio",
11                                         "sylvio@units.it",
12                                         Professor.BACKGROUND.INFORMATICA);
13     Seminar seminar = new Seminar("Machine Learning Overview", professor, 20);
14
15     char azione = ' ';
16     while(azione!='U'){
17         System.out.println("Qual è l'azione desiderata?");
18         System.out.println(" [I] - Iscrivere studenti");
19         System.out.println(" [E] - Elencare studenti");
20         System.out.println(" [C] - Cancelare studenti");
21         System.out.println(" [U] - Esci");
22         azione = (char)sc.next().charAt(0);
23         switch(azione){
24             case 'I':
25                 iscrivere(seminar);
26                 break;
27             case 'E':
28                 elencare(seminar);
29                 break;
30             case 'C':
31                 cancellare(seminar);
32                 break;
33             case 'U':
34                 return;
35             default:
36                 System.out.println("** azione sconosciuta?");
37                 azione = ' ';
38         }
39     }
40 }
41

```

```

barbon@barbon-PC:~/Scaricati/FI/uml$ java MainSeminar
Qual è l'azione desiderata?
[I] - Iscrivere studenti
[E] - Elencare studenti
[C] - Cancelare studenti
[U] - Esci

```

2) Incapsulamento

Esempi

```
42 public void iscrivere(Seminar seminar){
43     String nome, email;
44     System.out.println("-- Iscrizione studenti al Seminario --");
45     do{
46         System.out.println("Inserisci il nome dello studente:");
47         nome = sc.next();
48         System.out.println("Inserisci la mail dello studente:");
49         email = sc.next();
50         if(seminar.addStudent(new Student(nome,email))){
51             System.out.println("** successo");
52         }else{
53             System.out.println("** errore");
54         }
55         System.out.println("Vuoi aggiungere un altro studente [S/N]?");
56     }while(sc.next().equals("S"));
57 }
```

Qual è l'azione desiderata?

[I] - Iscrivere studenti
[E] - Elencare studenti
[C] - Cancelare studenti
[U] - Esci

I

-- Iscrizione studenti al Seminario --

Inserisci il nome dello studente:

Sylvio

Inserisci la mail dello studente:

sylvio@gmail.com

** successo

Vuoi aggiungere un altro studente [S/N]?

2) Incapsulamento Esempi

```
59     public void elencare(Seminar seminar){
60         System.out.println("-- Elenchi degli studenti del seminario --");
61         System.out.println("-- "+ seminar +" --");
62         Student[] students = seminar.listStudent();
63         for(int i = 0; i<students.length; i++){
64             if(students[i]!=null){
65                 System.out.println((i+1)+" - "+students[i].toString());
66             }
67         }
68     }
69 }
```


2) Incapsulamento

Esempi

```
70 public void cancellare(Seminar seminar){
71     String nome, email;
72     System.out.println("-- Cancellare studenti al Seminario --");
73     do{
74         System.out.println("Qual è il nome dello studente?");
75         nome = sc.next();
76         System.out.println("Qual è la mail dello studente?");
77         email = sc.next();
78         if(seminar.remStudent(new Student(nome,email))){
79             System.out.println("** successo");
80         }else{
81             System.out.println("** errore");
82         }
83         System.out.println("Vuoi cancellare un altro studente [S/N]?");
84     }while(sc.next().equals("S"));
85 }
86 }
```