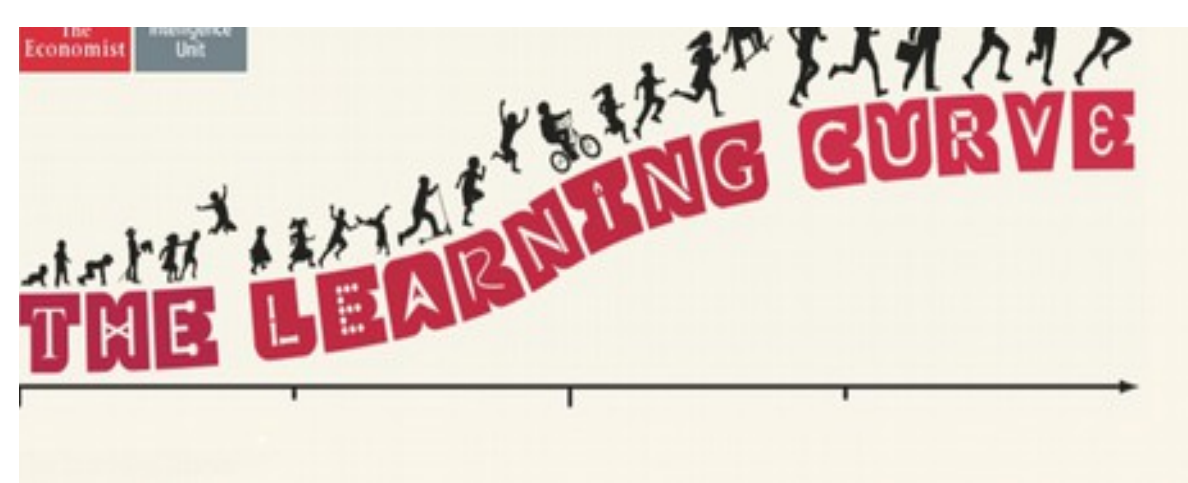


Programmazione ad oggetti

JAVA

a.f. 2017-18

La conoscenza che non entra nella carne
è solo rumore!



Argomenti

- Analisi e programmazione - Algoritmi
- Oop - Programmazione Orientata agli Oggetti
- Pattern, MVC
- Java 8
- JavaScript - EcmaScript 6
-

I principi della OOP

- Definire nuovi tipi di dati.
- Incapsulare i valori e le operazioni.
- Riusare il codice esistente (ereditarietà).
- Supportare il polimorfismo.

ADT creare nuovi tipi

- Dati
- Operazioni (metodi)
 - *che agiscono sui dati, leggendoli o scrivendoli*

ADT tramite l'incapsulamento

La classe consente di implementare gli ADT attraverso il meccanismo di incapsulamento:

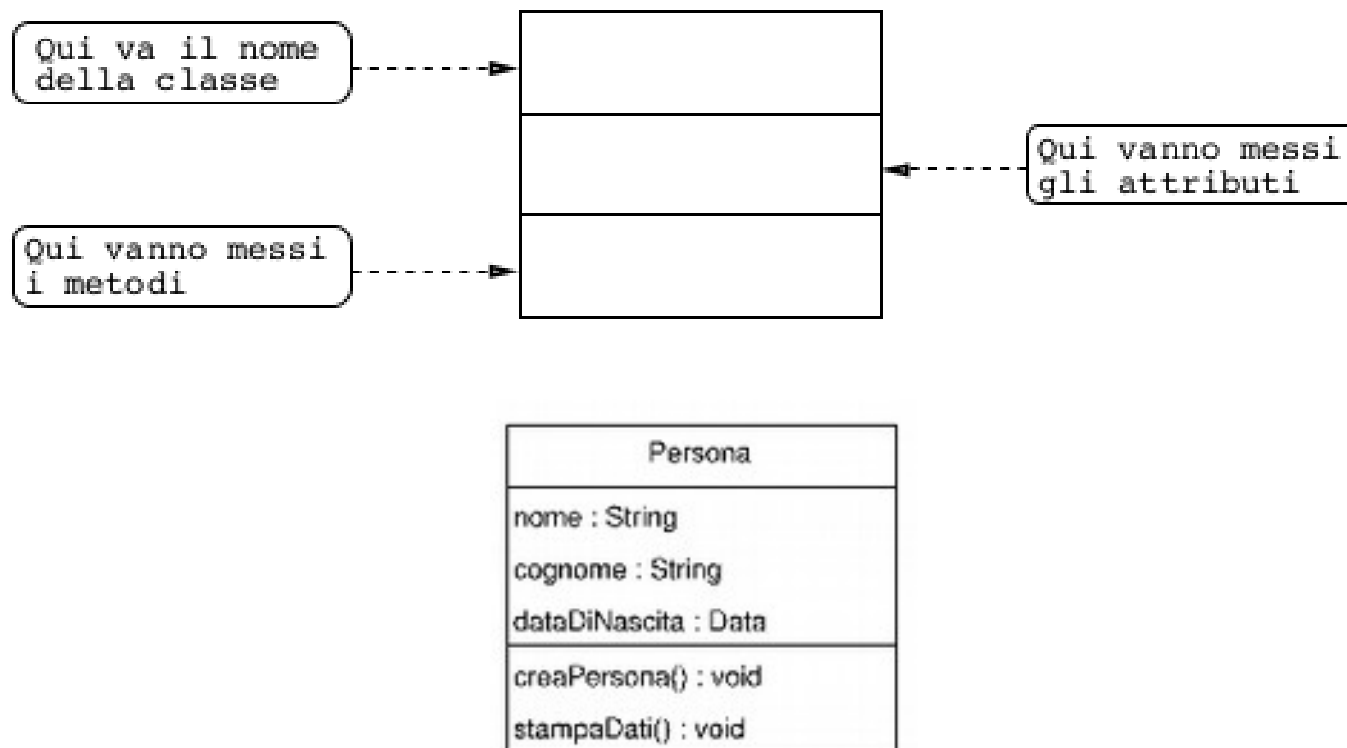
- i **dati** devono rimanere privati insieme all'implementazione
- solo l'**interfaccia** delle operazioni è resa pubblica all'esterno della classe.

La descrizione di una classe deve elencare

- i dati (o **attributi**)
 - contengono le informazioni di un oggetto;
- le operazioni (o **metodi**)
 - consentono di leggere/scrivere gli attributi di un oggetto;

UML - Unified Modeling Language

- In UML una classe si rappresenta così:



L'oggetto

- un oggetto è una istanza di una classe.
- un oggetto deve essere conforme alla descrizione di una classe.
- un oggetto è contraddistinto da:
 - 1. attributi;
 - 2. metodi;
 - 3. identità;

Persona
nome : String cognome : String dataDiNascita : Data
creaPersona() : void stampaDati() : void

L'oggetto

un oggetto non deve mai manipolare direttamente i dati di un altro oggetto

la **classe** è una entità **statica** cioè a tempo di compilazione;

l'oggetto è una entità **dinamica** cioè a tempo di esecuzione (*run time*);

Relazioni fra le classi

- **uso**: una classe può usare oggetti di un'altra classe;
- **aggregazione**: una classe può avere oggetti di un'altra classe;
- **ereditarietà**: una classe può estendere un'altra classe.

Uso

L'uso o associazione è la relazione più semplice che intercorre fra due classi.

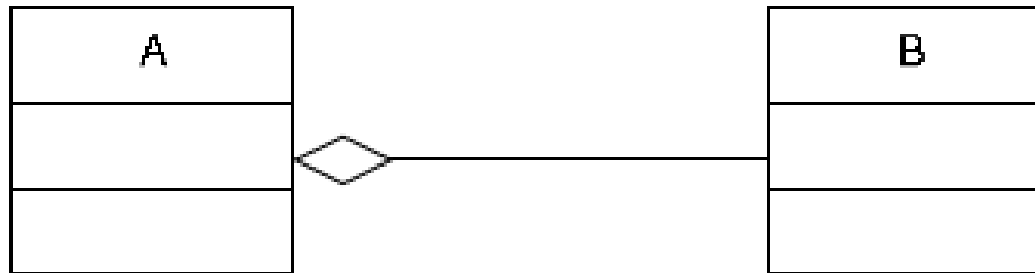
Per definizione diciamo che una classe A usa una classe B se:

- un metodo della classe A invia messaggi agli oggetti della classe B , oppure
- un metodo della classe A crea, restituisce, riceve oggetti della classe B .



Aggregazione

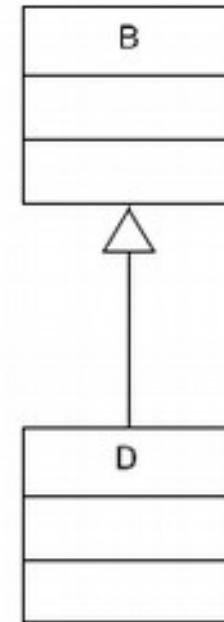
- una classe A **aggrega** oggetti di una classe B quando la classe A **contiene** oggetti della classe B
- è un caso speciale della relazione di uso
- relazione **has-a** (ha-un)



il rombo è attaccato alla classe che contiene l'altra

Ereditarietà

- **riuso** del codice
- classe derivata o **sottoclasse**
- classe base o **superclasse**
- Relazione **is-a** (è-un)

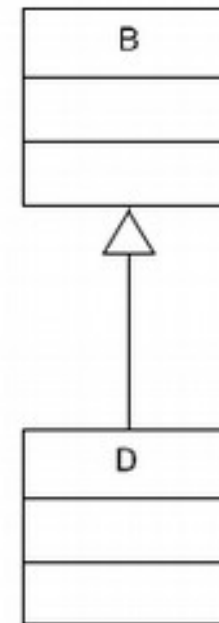


Ereditarietà: si ottiene il riuso del codice

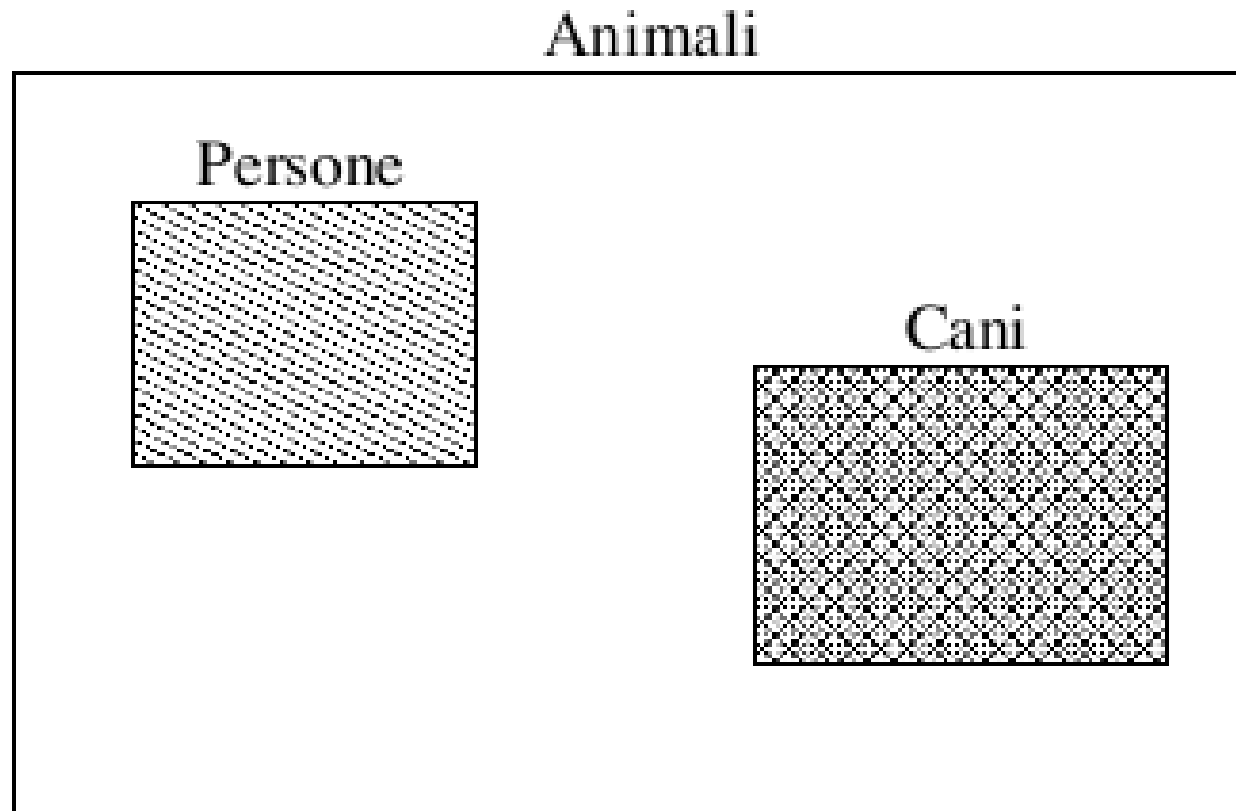
Consideriamo una **classe base B** che ha un metodo $f(\dots)$ ed una **classe derivata D** che eredita da B .

La classe D può usare il metodo $f(\dots)$ in tre modi:

- lo **eredita**: quindi $f(\dots)$ può essere usato come se fosse un metodo di D ;
- lo **riscrive** (*override*): cioè si dà un nuovo significato al metodo riscrivendo la sua implementazione nella classe derivata, in modo che tale metodo esegua una azione diversa;
- lo **estende**: cioè richiama il metodo $f(\dots)$ della classe base ed aggiunge altre operazioni.



la classe Persona deriva da una classe molto più grande, cioè la classe degli Animali

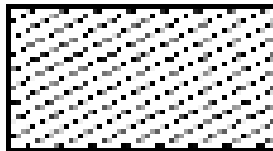


gerarchia di
classi

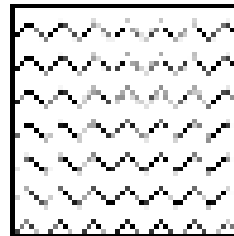
Animali

Mammiferi

Cani

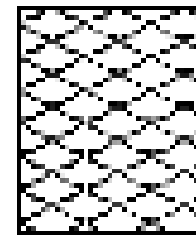


Esseri
Umani

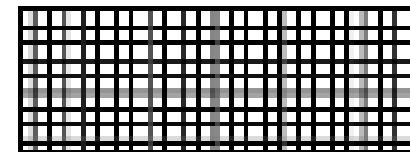


Insetti

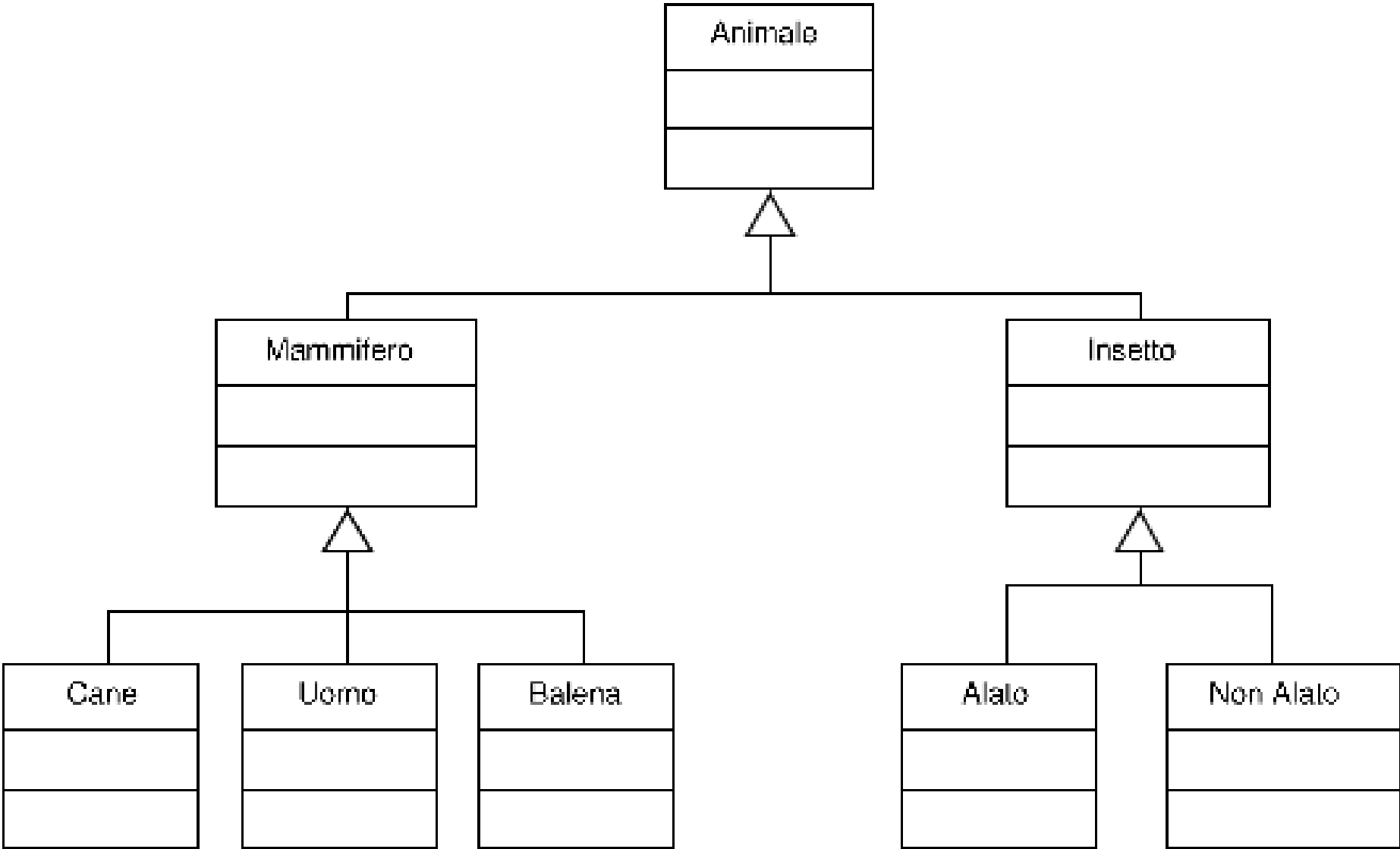
Alati



Non Alati



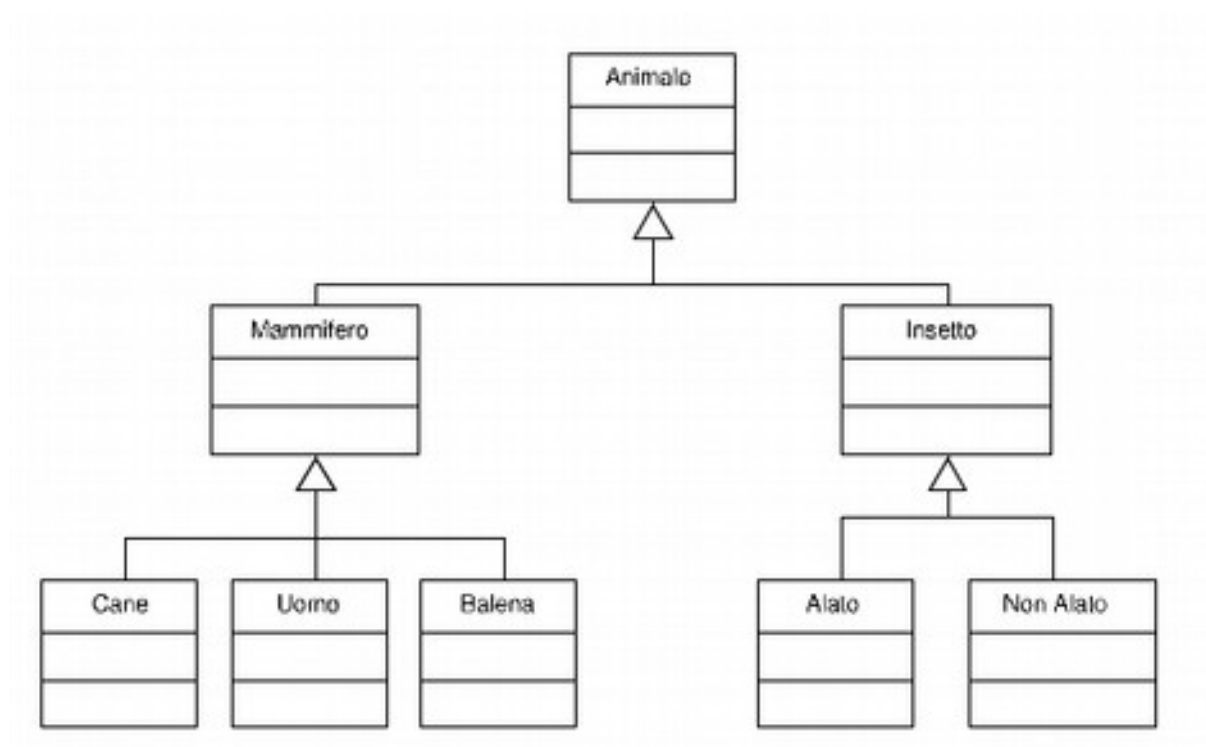
gerarchia di
classi



Classi astratte

esaminiamo i metodi della classe base Animale

- comunica(...), mangia(), siMuove(), ...



classe astratta

- Una classe che ha almeno un metodo astratto si dice **classe astratta**
- può anche contenere dei metodi non astratti
- è troppo generica per essere istanziata
- usata come un **contenitore di comportamenti** (operazioni) comuni che ogni classe derivata sa come implementare

Quando usare l'ereditarietà

- Usare l'ereditarietà solo quando il legame fra la classe base e la classe derivata è per sempre, cioè dura per tutta la vita degli oggetti, istanze della classe derivata.
- Se tale legame non è duraturo è meglio usare l'aggregazione al posto della specializzazione.

Polimorfismo

- La parola polimorfismo deriva dal greco e significa letteralmente molte forme.
- Nella OOP tale termine si riferisce ai **metodi**: per definizione, il polimorfismo è la capacità di un oggetto, la cui classe fa parte di una gerarchia, di chiamare la versione corretta di un metodo.
- Quindi il polimorfismo è necessario quando si ha una gerarchia di classi.

Programma JAVA (OCA)

- 1 Java Building Blocks
- 2 Operators and Statements
- 3 Core Java APIs
- 4 Methods and Encapsulation
- 5 Class Design
- 6 Exceptions

Programma JAVA (OCP)

- 1 Advanced Class Design
- 2 Design Patterns and Principles
- 3 Generics and Collections
- 4 Functional Programming
- 5 Dates, Strings, and Localization
- 6 Exceptions and Assertions
- 7 Concurrency
- 8 IO
- 9 NIO.2
- 10 JDBC

Pseudo linguaggio

Area del rettangolo

Inizio

Leggi (Base)

Leggi (Altezza)

Area = Base * Altezza

Scrivi (Area)

fine

Un algoritmo è una descrizione dettagliata, per **passi elementari** successivi, di una strategia utile per risolvere un determinato problema.

- Ogni algoritmo è un insieme **finito** di passi e deve **terminare** dopo un numero finito di iterazioni.

- Nella progettazione di un algoritmo il programmatore inizia a porsi problemi relativi alla **rappresentazione delle informazioni** che deve essere **efficiente** (senza sprechi inutili) ed **efficace** (non si deve perdere traccia di dati importanti).

- Naturalmente l'aspetto fondamentale è la progettazione di un **algoritmo efficiente**.

Teorema fondamentale della programmazione strutturata

Teorema fondamentale della programmazione strutturata
Jacopini e Bohm

Che ogni programma può essere codificato utilizzando tre strutture
Fondamentali

- | Sequenziale
- | Iterativa
- | Condizionale

OOP

- | Linguaggi basso livello
- | Linguaggi alto livello
 - | Procedurale, imperativo, funzionale: C
 - | Orientato agli oggetti:
 - | Mantiene compatibilità con C: C++,
 - | Non compatibile con C: Java

classe Automobile

n_ruote = 4;
colore = blu;

classe Motocicletta

n_ruote = 2;

Principi

- | Definire nuovi dati
- | Incapsulare valori, operazioni
- | Riutilizzare il codice sorgente (ereditarietà)
- | Oggetto deve avere diverse forme (polimorfismo)

Cos'è un tipo primitivo?

Numeri, caratteri e booleans

memorizzato in una zona di memoria più rapida

i nomi dei tipi sono minuscoli

String non è un tipo primitivo

Dichiarare variabili primitive

java è un linguaggio tipato

ogni variabile deve dichiarare il suo tipo

```
int miaVar = 5;
```

tipo identificatore (camelCase) assegnazione valore

tipi primitivi numerici

tipi primitivi numerici

Data Type	Bits	Minimum	Maximum
byte	8	-128	127
short	16	-32,768	32,767
int	32	-2,147,483,648	2,147,483,647
long	64	-9.22337E+18	9.22337E+18
float	32	See the docs	
double	64	See the docs	

Classi wrapper (helper)

java include wrapper classes per ogni tipo primitivo
le classi helper forniscono strumenti per convertire e
formattare i dati

p.es

Data Type	Helper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float

Helper Class Example

- `java.lang.Double` supports double values.

```
double doubleValue = 156.5d;  
byte byteValue = Double.byteValue(doubleValue);  
int intValue = Double.intValue(doubleValue);  
float floatValue = Double.floatValue(doubleValue);  
String stringValue = Double.toString(doubleValue);
```


Il valore di default dei tipi primitivi è 0

```
int myInt;
```

```
//myInt vale 0
```

Ex PrimitiveNumbers

```
package com.example.java;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        byte b = 1;
```

```
        short sh = 1;
```

```
        int i = 1;
```

```
        long l = 1L;
```

```
        float f = 1f;
```

```
        double d = 1d;
```

```
        System.out.println("Byte: " + b);
```

```
        System.out.println("Short: " + sh);
```

```
        System.out.println("Int: " + i);
```

```
        System.out.println("Long: " + l);
```

```
        System.out.println("Float: " + f);
```

```
        System.out.println("Double: " + d);
```

```
package com.example.java;

public class MaxValues {

    public static void main(String[] args) {

        byte b = 127;
        System.out.println("Byte: " + b);

        if (b < Byte.MAX_VALUE) {
            b++;
        }

        System.out.println("Byte: " + b);

    }

}
```

- lavorare con le valute (double) 4.03

-

console:

la copia di un tipo primitivo è sempre per valore

cast

```
package com.example.java;
```

```
import java.math.BigDecimal;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        double value = .012;
```

```
        double pSum = value + value + value;
```

```
        System.out.println("Sum of primitives: " + pSum);
```

```
        String strValue = Double.toString(value);
```

```
        System.out.println("strValue: " + strValue);
```

```
        BigDecimal bigValue = new BigDecimal(strValue);
```

```
        BigDecimal bSum = bigValue.add(bigValue).add(bigValue);
```

```
        System.out.println("Sum of BigDecimals: " + bSum.toString());
```

```
}
```

```
public static void main(String[] args) {  
  
    int intValue1 = 56;  
    int intValue2 = intValue1;  
    System.out.println("The 2nd value is " + intValue2);  
  
    long longValue1 = intValue1;  
    System.out.println("The long value is " + longValue1);  
  
    short shortValue1 = (short) intValue1;  
    System.out.println("The short value is " + shortValue1);  
  
    int intValue3 = 1024;  
    byte byteValue = (byte) intValue3;  
    System.out.println("The byte value is " + byteValue);  
  
    double doubleValue = 3.999999d;  
    int intValue4 = (int) doubleValue;  
    System.out.println("Double to int: " + intValue4);  
  
}
```

cast (widening / narrowing)

- lavorare con gli operatori matematici

```
public static void main(String[] args) {
```

```
    int intValue1 = 56;
```

```
    int intValue2 = 42;
```

```
    int result1 = intValue1 + intValue2;
```

```
    System.out.println("Addition: " + result1);
```

```
    int result2 = intValue1 - intValue2;
```

```
    System.out.println("Subtraction: " + result2);
```

```
    int result3 = intValue1 * intValue2;
```

```
    System.out.println("Multiplication: " + result3);
```

```
    double result4 = (double) intValue1 / intValue2;
```

```
    System.out.println("Division: " + result4);
```

```
    double result5 = (double) intValue1 % intValue2;
```

```
    System.out.println("Remainder: " + result5);
```

```
    double doubleValue = -3.99999;
```

```
    long rounded = Math.round(doubleValue);
```

```
    System.out.println("Rounded: " + rounded);
```

```
    double absValue = Math.abs(doubleValue);
```

```
    System.out.println("Absolute: " + absValue);
```

```
}
```

shift + f1 per la doc sulla classe math (appoggiando il mouse sul nome della classe)

- lavorare con booleans

Base

```
public static void main(String[] args) {  
  
    boolean b1 = true;  
    boolean b2 = false;  
  
    System.out.println("The value of b1 is: " + b1);  
    System.out.println("The value of b2 is: " + b2);  
}
```

di default è false, prova con una var static non locale (quelle locali devi per forza inizializzarle)

```
static boolean bDef;  
public static void main(String[] args) {  
  
    boolean b1 = true;  
    boolean b2 = false;  
  
    System.out.println("The value of b1 is: " + b1);  
    System.out.println("The value of b2 is: " + b2);  
    System.out.println("The value of bDef is: " + bDef);  
}
```

diverso da boolean

```
boolean b3 = !b1;  
System.out.println("The value of b3 is: " + b3);  
  
int i1 = 0;  
boolean b4 = (i1 != 0);  
System.out.println("The value of b4 is: " + b4);
```

per parsare una stringa uso la classe wrapper

```
String sBoolean = "true";  
boolean parsed = Boolean.parseBoolean(sBoolean);  
System.out.println("Parsed: " + parsed);
```

- lavorare con char

È un tipo primitivo, non una stringa complessa

usi gli apici singoli

la sua classe wrapper fornisce dei metodi di utilità

```
public static void main(String[] args) {
```

```
    char c1 = '1';
```

```
    char c2 = '2';
```

```
    char c3 = '3';
```

```
    System.out.println("Char 1: " + c1);
```

```
    System.out.println("Char 2: " + c2);
```

```
    System.out.println("Char 3: " + c3);
```

```
    char dollarSign = '\u0024';
```

```
    System.out.println("Currency: " + dollarSign);
```

```
    char a1 = 'a';
```

```
    char a2 = 'b';
```

```
    char a3 = 'c';
```

```
    System.out.print(Character.toUpperCase(a1));
```

```
    System.out.print(Character.toUpperCase(a2));
```

```
    System.out.println(Character.toUpperCase(a3));
```

Assignment and Math Operators

```
int intValue;
```

```
int intValue = 10;
```

↑
Simple assignment

Assignment and simple math

```
int newValue = intValue + 5; 15
```

```
int newValue = intValue - 5; 5
```

```
int newValue = intValue * 5; 50
```

```
int newValue = intValue / 5; 2
```

```
int newValue = intValue % 5; 0
```


More Assignment and Math

```
int intValue = 10;
```

↑
Initial value

Incrementing and decrementing

```
intValue++;
```

11

```
intValue--;
```

9

Other assignments

```
intValue += 5;
```

15

```
intValue -= 5;
```

5

```
intValue *= 5;
```

50

```
intValue /= 5;
```

2

Postfix vs. Prefix Incrementing

```
int intValue = 10;
```



Initial value

Postfix

```
System.out.println(intValue ++);
```

Output: 10
New value: 11

Prefix

```
System.out.println(++ intValue);
```

Output: 11
New value: 11

Equality Operators

Equality

```
if (this == that) {  
    System.out.println("They match!");  
}
```

Inequality

```
if (this != that) {  
    System.out.println("No match!");  
}
```

Comparing Values

Operator	Purpose
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
instanceof	Class membership

```
String s = "Hello";  
if (s instanceof java.lang.String) {  
    System.out.println("s is a String");  
}
```

Comparing Strings

```
String s1 = "Hello";  
String s2 = "Hello";  
if (s1.equals(s2)) {  
    System.out.println("They match!");  
}  
else {  
    System.out.println("No match!");  
}
```

They match!

devi usare equals per confrontare gli oggetti

Logical Operators

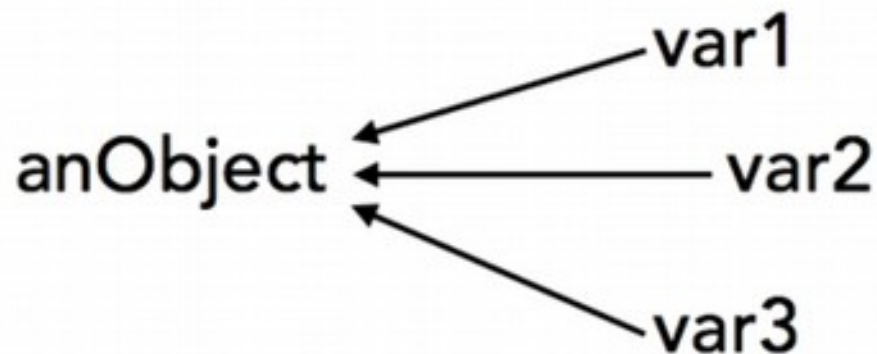
Operator	Purpose
&&	And
	Or
?=	Ternary (short-hand for if-then)

```
String s = condition ? trueValue : valoreFalse ;
```




oggetti e variabili reference

Java Classes Define Objects

- An object is an instance of a class.
- Nonprimitive variables are references to objects.
- Objects can have multiple references.



A Class Instantiating Itself

```
public class ClothingItem {  
    public String type;  Instance variable  
  
    public static void main(String[] args) {  
        ClothingItem item = new ClothingItem();  Instantiation  
        item.type = "Hat";  
        item.displayItem();  
    }  
  
    private void displayItem()  Instance method {  
        System.out.println("This is a " + this.type);  
    }  
}
```

A String Is an Object

- String values are instances of `java.lang.String`.

```
String string1 = new String("Hello!");
```

```
String string2 = "Hello!";
```

Other Characteristics of Strings

- A string is an array of characters.

A String object

"Hello!"

An array of Char values

H	e	l	l	o	!
---	---	---	---	---	---

Other Characteristics of Strings

- String objects are immutable.
- Resetting the value of a String creates a new object.

Char Array to a String

```
public class Main {  
    public static void main(String[] args) {  
        char[] chars = {'H','e','l','l','o','!'};  
        String s = new String(chars);  
        System.out.println(s);  
    }  
}
```

Shift + F1 per vedere la classe String

```
public static void main(String[] args) {  
  
    String s1 = "This is a String!";  
    System.out.println(s1);  
  
    String s2 = new String("This is also a String!");  
    System.out.println(s2);  
  
    String s3 = "Shirt size: ";  
    String s4 = "M";  
    String s5 = s3 + s4 + ", Qty: " + 4;  
    System.out.println(s5);  
  
    char[] chars = {'H', 'e', 'l', 'l', 'o'};  
    String s6 = new String(chars);  
    System.out.println(s6);  
  
    char[] chars2 = s6.toCharArray();  
    for (char c : chars2) {  
        System.out.println(c);  
    }  
}
```

...

```
import java.text.NumberFormat;
```

...

```
public static void main(String[] args) {
```

```
    int intValue = 42;
```

```
    String fromInt = Integer.toString(intValue);
```

```
    System.out.println(fromInt);
```

```
    boolean boolValue = true;
```

```
    String fromBool = Boolean.toString(boolValue);
```

```
    System.out.println(fromBool);
```

```
    long longValue = 10_000_000;
```

```
    NumberFormat formatter = NumberFormat.getInstance();
```

```
    String formatted = formatter.format(longValue);
```

```
    System.out.println(formatted);
```

```
}
```

convertire tipi primitivi a stringa

per dare un formato devo importare la classe

NumberFormat

creo 3 oggetti distinti
posso usare invece StringBuilder

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        String str1 = "Hello";
        String str2 = "World";
        String str3 = str1 + ", " + str2 + "!";
        System.out.println(str3);
```

anche in forma compatta

```
    StringBuilder sb = new StringBuilder("Hello")
        .append(", ")
        .append("World")
        .append("!");
    System.out.println(sb);
```


uso la classe Scanner per catturare l'input e memorizzarlo in StringBuilder

```
Scanner scanner = new Scanner(System.in);
System.out.print("Enter value: ");
String input = scanner.nextLine();
System.out.println(input);

sb.delete(0, sb.length());
for (int i = 0; i < 3; i++) {
    input = scanner.nextLine();
    sb.append(input + "\n");
}

System.out.println(sb);

}

}
```

anche in forma compatta

la inserisco in un ciclo

```
public static void main(String[] args) {  
  
    String str1 = "Hello";  
    String str2 = "World";  
    String str3 = str1 + ", " + str2 + "!";  
    System.out.println(str3);  
  
    StringBuilder sb = new StringBuilder("Hello"  
        .append(", ")  
        .append("World")  
        .append("!");  
    System.out.println(sb);  
  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Enter value: ");  
    String input = scanner.nextLine();  
    System.out.println(input);  
  
    sb.delete(0, sb.length());  
    for (int i = 0; i < 3; i++) {  
        input = scanner.nextLine();  
        sb.append(input + "\n");  
    }  
  
    System.out.println(sb);  
}
```

Comparare le Stringhe, occhio che sono oggetti immutabili
il compilatore verifica che esista e in caso positivo crea
un reference alla string esistente. Vd esempio str4 e 5

```
String str1 = "Hello";  
String str2 = "Hello";  
  
if (str1 == str2) {  
    System.out.println("They match!");  
} else {  
    System.out.println("They don't match!");  
}  
  
String str3 = "hello";  
if (str1 == str3) {  
    System.out.println("They match!");  
} else {  
    System.out.println("They don't match!");  
}  
  
String part1 = "Hello ";  
String part2 = "World";  
String str4 = part1 + part2;  
String str5 = "Hello World";  
if (str4 == str5) {  
    System.out.println("They match!");  
} else {  
    System.out.println("They don't match!");  
}
```

avrei dovuto usare equals o equalsIgnoreCase

```
String part1 = "Hello ";  
String part2 = "WORLD";  
String str4 = part1 + part2;  
String str5 = "Hello World";  
if (str4.equalsIgnoreCase(str5)) {  
    System.out.println("They match!");  
} else {  
    System.out.println("They don't match!");  
}
```

ancora sulla formattazione dei numbers
posso anche utilizzare la classe Locale per la localizz

```
import java.text.NumberFormat;  
import java.util.Locale;  
  
public class Main {  
    public static void main(String[] args) {  
        Locale locale = new Locale("da", "DK");  
  
        double doubleValue = 1_234_567.89;  
  
        NumberFormat numF = NumberFormat.getInstance(locale);  
        System.out.println("Number: " + numF.format(doubleValue));  
  
        NumberFormat curF = NumberFormat.getCurrencyInstance(locale);  
        System.out.println("Currency: " + curF.format(doubleValue));  
  
        NumberFormat intF = NumberFormat.getIntegerInstance();  
        System.out.println("Integer: " + intF.format(doubleValue));  
    }  
}
```


Altri metodi sulle stringhe

```
public static void main(String[] args) {  
  
    String s1 = "Welcome to California!";  
    System.out.println("Length of string: " + s1.length());  
  
    int position = s1.indexOf("California");  
    System.out.println("Position of substring: " + position);  
  
    String sub = s1.substring(11);  
    System.out.println(sub);  
  
    String s2 = "Welcome!";  
    int len1 = s2.length();  
    System.out.println(len1);  
    String s3 = s2.trim();  
    System.out.println(s3.length());  
}
```

una pillola di date e time

```
import java.util.Date;

public class Main {

    public static void main(String[] args) {

        Date d = new Date();
        System.out.println(d);

        Gre
        c GregorianCalendar (java.util)
    } c Gregorian (sun.util.calendar)
```

posso usare `GregorianCalendar` per avere più metodi

```
import java.text.DateFormat;
import java.util.Date;
import java.util.GregorianCalendar;

public class Main {

    public static void main(String[] args) {

        Date d = new Date();
        System.out.println(d);

        GregorianCalendar gc = new GregorianCalendar(2009, 1, 28);
        gc.add(GregorianCalendar.DATE, 1);
        Date d2 = gc.getTime();
        System.out.println(d2);

        DateFormat df = DateFormat.getDateInstance(DateFormat.FULL);
        System.out.println(df.format(d2));
    }
}
```


con java 8 posso usare locale e formattare in diversi modi

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Date;
import java.util.GregorianCalendar;

public class Main {

    public static void main(String[] args) {

        Date d = new Date();
        System.out.println(d);

        GregorianCalendar gc = new GregorianCalendar(2009, 1, 28);
        gc.add(GregorianCalendar.DATE, 1);
        Date d2 = gc.getTime();
        System.out.println(d2);

        DateFormat df = DateFormat.getDateInstance(DateFormat.FULL);
        System.out.println(df.format(d2));

        LocalDateTime ldt = LocalDateTime.now();
        System.out.println(ldt);

        LocalDate ld = LocalDate.of(2009, 1, 28);
        System.out.println(ld);

        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("M/d/yyyy");
        System.out.println(dtf.format(ld));
    }
}
```

Semplice calcolatore

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a numeric value: ");
        String input1 = sc.nextLine();
        double d1 = Double.parseDouble(input1);

        System.out.print("Enter a numeric value: ");
        String input2 = sc.nextLine();
        double d2 = Double.parseDouble(input2);

        double result = d1 + d2;

        System.out.println("The answer is " + result);

    }

}
```

calcolo somma con perc interesse

```
double percentualeInteresse = 2.5; //percentuale di interesse annuale
double sommaDepositata = 1000; //in euro

//dopo un anno
sommaDepositata = sommaDepositata * (1 + percentualeInteresse/100);
System.out.println("Dopo un anno avrai: " + sommaDepositata + " euro");

//dopo un altro anno
sommaDepositata = sommaDepositata * (1 + percentualeInteresse/100);
System.out.println("Dopo un secondo anno avrai: " + sommaDepositata + " euro");

//e dopo un altro ancora ...
sommaDepositata = sommaDepositata * (1 + percentualeInteresse/100);
System.out.println("Dopo un anno avrai: " + sommaDepositata + " euro");
```

es. punti: localizza quadrante

```
//altro  
public int LocalizzaQuadrante()  
{  
    if (x>0 && y>0)  
        return 1;  
    else  
        if (x<0 && y>0)  
            return 2;  
        else  
            if (x<0 && y<0)  
                return 3;  
            else  
                if (x>0 && y<0)  
                    return 4;  
                else  
                    return 99;  
}  
}
```

Cos'è un oggetto

Un'istanza (esemplare) di una classe (tipo)

se immaginiamo una classe come uno stampo per biscotti

un biscotto è un oggetto

se lo compri in un negozio:

non è un dolce

non uno specifico tipo di dolce

è un oggetto reale: mi dia quel dolce!

definire una classe

una classe ha membri

dati

metodi

esempio stampante

dati

modello

produttore

carica fogli

colore inchiostro

metodi

stampante

accendi

spegni

mostra errori

cicli e condizioni, il flusso del programma

if else

```
int monthNumber = 13;

if (monthNumber >= 1 && monthNumber <= 3) {
    System.out.println("You're in Quarter 1");
} else if (monthNumber >= 4 && monthNumber <= 6) {
    System.out.println("You're in Quarter 2");
} else if (monthNumber >= 7 && monthNumber <= 9) {
    System.out.println("You're in Quarter 3");
} else if (monthNumber >= 10 && monthNumber <= 12) {
    System.out.println("You're in Quarter 4");
} else {
    System.out.println("That's an unknown month!");
}
```

switch

```
Scanner sc = new Scanner(System.in);
```

```
System.out.print("Enter a number: ");
```

```
String input = sc.nextLine();
```

```
int monthNumber = Integer.parseInt(input);
```

```
switch (monthNumber) {
```

```
    case 1:
```

```
        System.out.println("The month is January");
```

```
        break;
```

```
    case 2:
```

```
        System.out.println("The month is February");
```

```
        break;
```

```
    case 3:
```

```
        System.out.println("The month is March");
```

```
        break;
```

```
    default:
```

```
        System.out.println("You chose another month!");
```

```
}
```

da java 7 è possibile usare switch con le stringhe

```
import java.util.Scanner;

public class SwitchStrings {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String input = sc.nextLine();

        switch (input) {
            case "Jan":
                System.out.println("The month is 1");
                break;
            case "Feb":
                System.out.println("The month is 2");
                break;
            case "Mar":
                System.out.println("The month is 3");
                break;
            default:
                System.out.println("You chose another month!");
        }
    }
}
```


si possono usare i loop per scorrere array

```
String[] months = {"January", "February", "March",  
    "April", "May", "June",  
    "July", "August", "September",  
    "October", "November", "December"};
```

```
for (int i = 0; i < months.length; i++) {  
    System.out.println(months[i]);  
}
```

o scorrere all'inverso

```
for (int i = months.length-1; i >= 0; i--) {  
    System.out.println(months[i]);  
}
```

```
String[] months = {"January", "February", "March",  
    "April", "May", "June",  
    "July", "August", "September",  
    "October", "November", "December"};
```

foreach in Java

```
for (String month : months) {  
    System.out.println(month);  
}
```

```
String[] months = {"January", "February", "March",  
    "April", "May", "June",  
    "July", "August", "September",  
    "October", "November", "December"};
```

posso scorrere l'array anche con un while loop

```
int counter = 0;  
while (counter < months.length) {  
    System.out.println(months[counter]);  
    counter++;  
}
```

**oppure sposto la condizione di valutazione al fondo,
così il codice viene eseguito almeno una volta do...while**

```
int counter = 0;  
do {  
    System.out.println(months[counter]);  
    counter++;  
} while (counter < months.length);
```