Mo Tu We Th Fr Sa Su

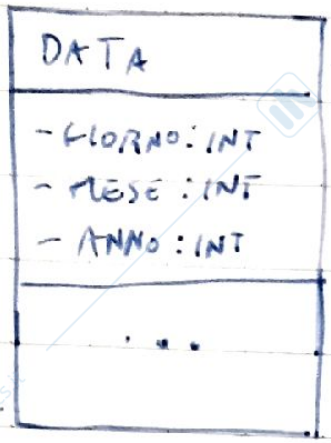
1

Ciò che troviamo dopo che usiamo lo strumento di traduz.

No. SISTEMI INF.

Date 4 · 12 · 19

CLASS DIAGRAMA → TRADUZIONE del codice → JAVA



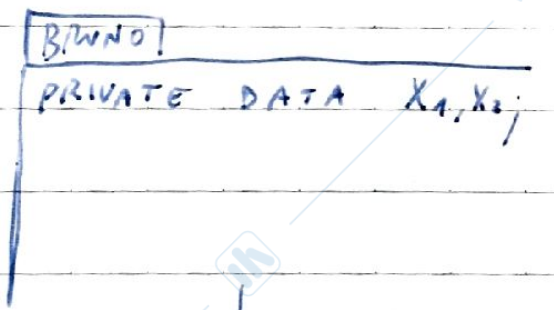

```
NOME CLASSE
PUBLIC CLASS DATA {
    PRIVATE INT GIORNO;
    PRIVATE INT MESE;      } ATTRIBUTI
    PRIVATE INT ANNO;
PUBLIC DATA ( ) {          } Costruttore Void
}
```

METODI → DEFINITI DALL'UTENTE

COSTRUTTORE

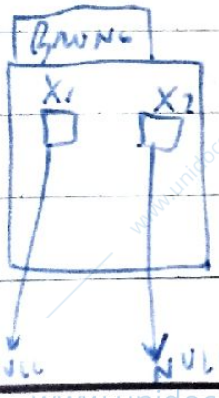
Supponiamo di avere un altro file in cui usiamo la classe precedente in un'altra classe

avrò

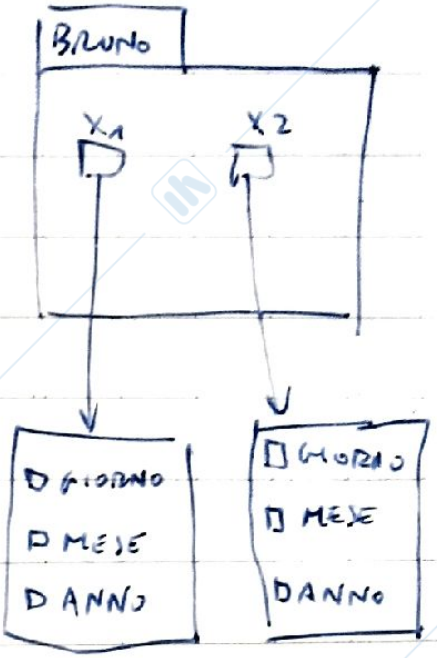


equivale a costruire puntatori a un dato DATA ma non ancora allocati (quindi non posso utilizzarli)

bisogna allocarli



```
BRUNO
PRIVATE DATA X1, X2;
X1 = NEW DATA ();
X2 = NEW DATA ();
```



è come fare una malloc

↓

il ruolo del COSTRUTTORE è fare questa malloc

Overload

In Java possiamo fare Overload ossia possiamo mandare a un metodo con lo stesso nome di un altro un input

Prospettiamo METODO COSTRUTTORE → STESSO NOME DEL COSTRUTTORE VOID

In PUBLIC DATA (INT AGIORNO, INT AMESE, INT AANNO){

THIS.GIORNO = AGIORNO;
THIS.MESE = AMESE;
THIS.ANNO = AANNO;

ATTRIBUTI DELLA CLASSE

PARAMETRI IN INGRESSO

NON è necessario mettere la A davanti ai parametri in ingresso

PER CHIAREZZA in generale agli attributi si scrive davanti THIS. → si riferisce all'oggetto stesso

3

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 4 12 19

Riassumiamo

CLASS DIAGRAM

DATA
- GIORNO : INT - MESE : INT - ANNO : INT
+ GET GIORNO() INT + SET GIORNO(GIORNO INT) VOID

AGGIUNGIAMO QUESTI METODI


```
PUBLIC INT GETGIORNO(){
    RETURN GIORNO;
}
PUBLIC VOID SETGIORNO(INT GIORNO){
    THIS.GIORNO = GIORNO;
}
```

Serve per evitare omonimia

In pratica riassumiamo come abbiamo creato la classe precedente

```
PUBLIC CLASS DATA{ ← DEFINIZIONE CLASSE
    PRIVATE INT GIORNO;
    PRIVATE INT MESE;
    PRIVATE INT ANNO;   ← DEF. ATTRIBUTI
    PUBLIC DATA(){ → COSTANTTORE VOID
    }
    PUBLIC DATA(INT GIORNO, INT MESE, INT ANNO){
        THIS.GIORNO = GIORNO;
        THIS.MESE = MESE;
        THIS.ANNO = ANNO;
    }
```

COSTANTTORE

```
PUBLIC INT GETGIORNO (){
        RETURN GIORNO;
}
PUBLIC VOID SETGIORNO (INT GIORNO){
        THIS.GIORNO=GIORNO;
}
:
}
```

METODI

Costruiamo adesso un'altra classe

```
PUBLIC CLASS USER {
    PRIVATE DATA X1;
    PRIVATE DATA X2;
        :
    PUBLIC VOID MAIN(){
        X1 = NEW DATA(1);
        X2 = NEW DATA(4,12,2019);
        X1.SETGIORNO(X2.GETMESE());
    }
```

ATTRIBUT.

COSTRUTTORE

OGGETTI

METODI

Passi: ① ② ③

5

Mo Tu We Th Fr Sa Su

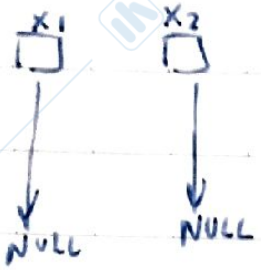
No. SISTEMI INF.

Date 4 12 19

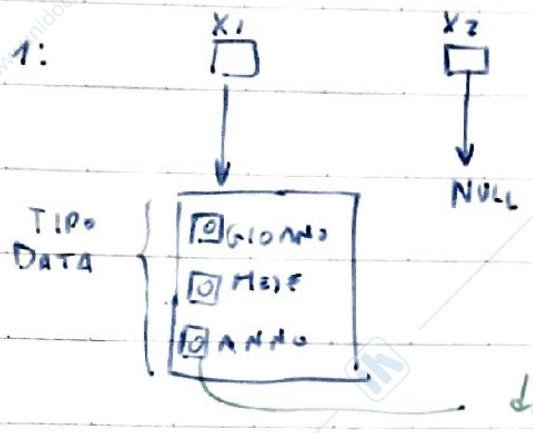
graficamente nel metodo MAIN

Prima di entrare nel main

Passo 0:

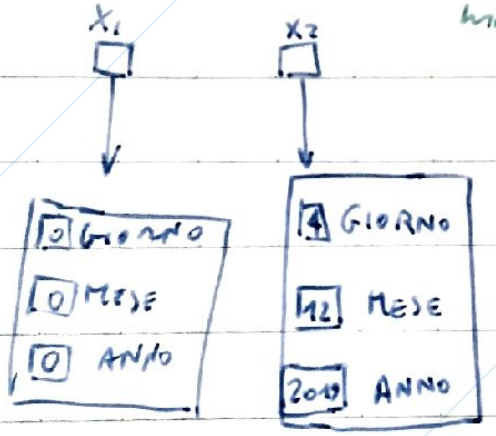


Passo 1:

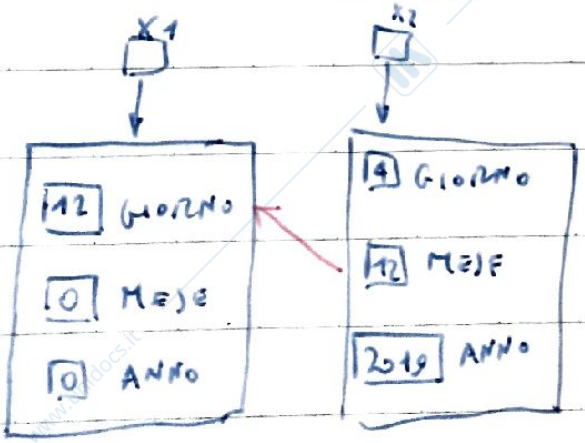


di solito gli interi vengono messi a 0 di default

Passo 2:



Passo 3:



6

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 4 12 19

STATIC

Riprendiamo le classi create in precedenza

DATA	USER
- ATTRIBUTI	- ATTRIBUTI
- METODI	- METODO PROVA

per essere eseguito ci dovrebbe un oggetto per il metodo

per fare

X = NEW DATA()

X. ...

es.

al posto di fare X.MAIN();

faccio direttamente

USER.MAIN();

si fa con

in dipendente dalla generazione di un oggetto

STATIC

ATTRIBUTI

METODI

In generale non possiamo usare i metodi o gli attributi di una classe per questo facciamo un oggetto

STANDARD, NON STATIC

CLASSE { ATTRIBUTI, METODI

OGGETTO → e poi usiamo ciò che è nella classe

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

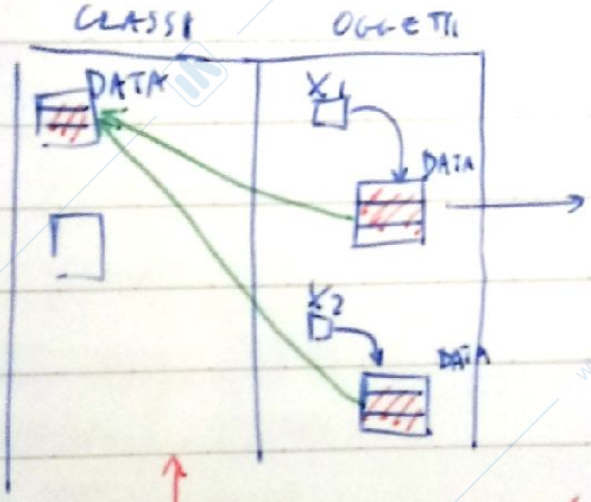
Date 4 · 12 · 19

Quindi nell'esempio dove usavamo main non abbiamo dovuto allocare nessun oggetto e fare poi Oggetto. MAIN

↓

Per cui avremmo dovuto definire una classe

Per avere un'idea più chiara possiamo immaginare di avere oggetti e classi



quando chiamiamo un metodo lo chiamiamo dalla "copia" della classe a cui punta l'oggetto (lavorando sulla classe 'copia')

Se usiamo un metodo Static lo puntiamo direttamente dalla classe

Un attributo STATIC è come se fosse un puntatore →

↳ Se lo modifichiamo nella classe DATA per es. si modifica anche l'attributo per X_1 e X_2

8

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 4 · 12 · 19

es. Testo:

```
PUBLIC CLASS CLASSEA {
    PUBLIC STATIC INT N;
    PRIVATE INT VAL;
    PRIVATE CLASSEA X;
  PUBLIC CLASSEA () {
        VAL = 0;
        N = 16;
        X = THIS;
  }
  PUBLIC CLASIEA (CLASSEA x){
      THIS. X = X;
      VAL = X. GET VAL() + 1;
      CLASSEA.N = CLASSEA.N/2;
  }
  PUBLIC INT GET VAL() {
      RETURN VAL;
  }
  PUBLIC CLASSEA GET X () {
      IF (VAL == 0)
          RETURN THIS.X;
      ELSE
          RETURN X. GETX();
  }
```

9

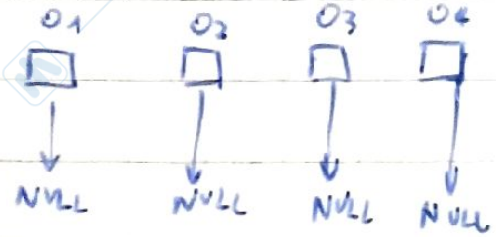
No. SISTEMI INF.

Mo Tu We Th Fr Sa Su

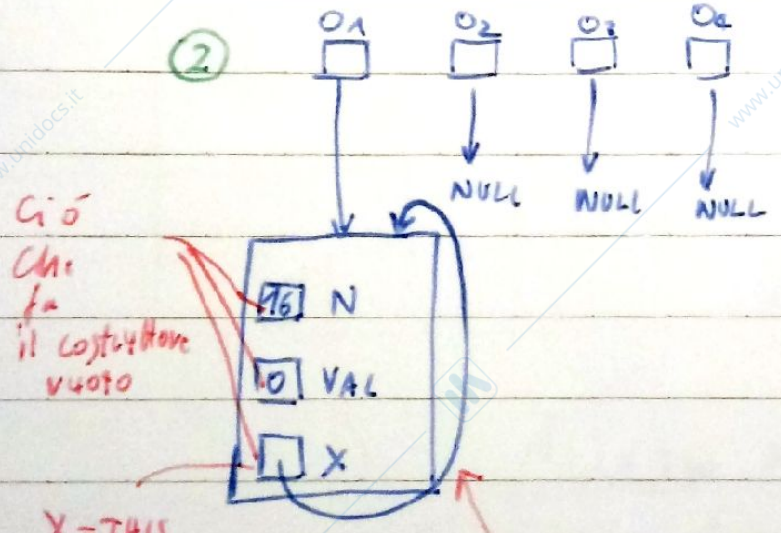
Date 4 · 12 · 19

```
PUBLIC STATIC VOID MAIN ( STRING[ ] ARGS) {
    (1) CLASSE A  O1,O2,O3,O4;
    (2) O1 = NEW CLASSEA ();
    (3) O2 = NEW CLASSEA (O1);
    (4) O3 = NEW CLASSEA (O2);
    (5) O4 = O3 . GETx ();
}
}
```

↳ Passi (1)



(2)



Ciò che fa il costruttore vuoto

X = THIS.

↓

devo guardare la d.t. di X — X è un attributo di CLASSEA

X punta alla stessa classe

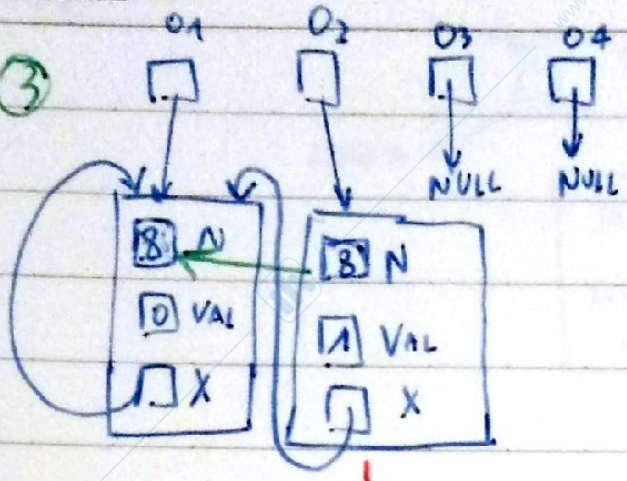
10

Mo Tu We Th Fr Sa Su

No. SISTEMI INF.

Date 4 · 12 · 19

3



→ abbiamo generato il codice del costruttore che riceve in ingresso una classe A dell'oggetto

Quando abbiamo

VAL = X.GETVAL() + 1 ≥ 1

O1

sostituisce VAL (=0)

Quando abbiamo

THIS.X = X

è l'X nel costruttore è O1 in ingresso al metodo del costruttore

è l'ultima applicazione di X fatta

Quando abbiamo

CLASSEAN = CLASSE.N[2]

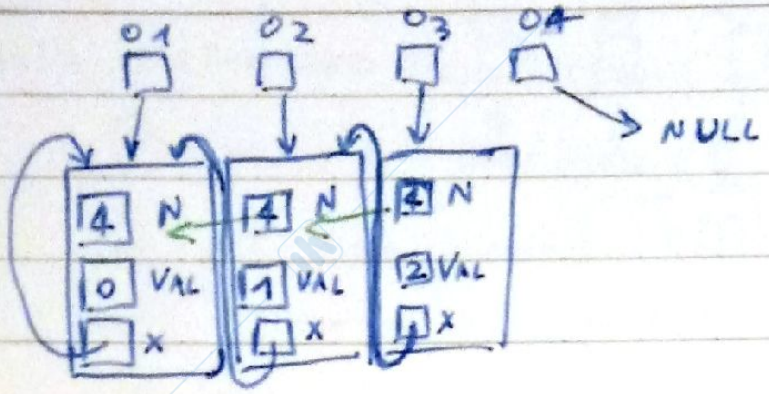
→ N è un attributo STATICO (→ cambia N in O1)

11

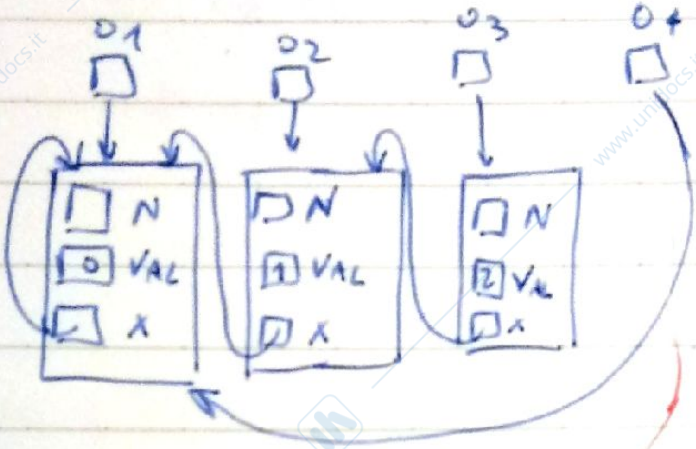
No. SISTEMI INF.

Date 4 · 12 · 19

④



⑤



dobbiamo prendere anche con:- il metodo GET.X() di O3

Abbiamo

IF (VAL == 0)

RETURN THIS.X

Siamo qui (poiché VAL in O3 = 2) → ELSE

RETURN X.GET.X();

in questo caso è O2

guardiamo GET.X in O2

VAL ≠ 0 in O2

guardiamo per la stessa condizione in O1 (X = O2)