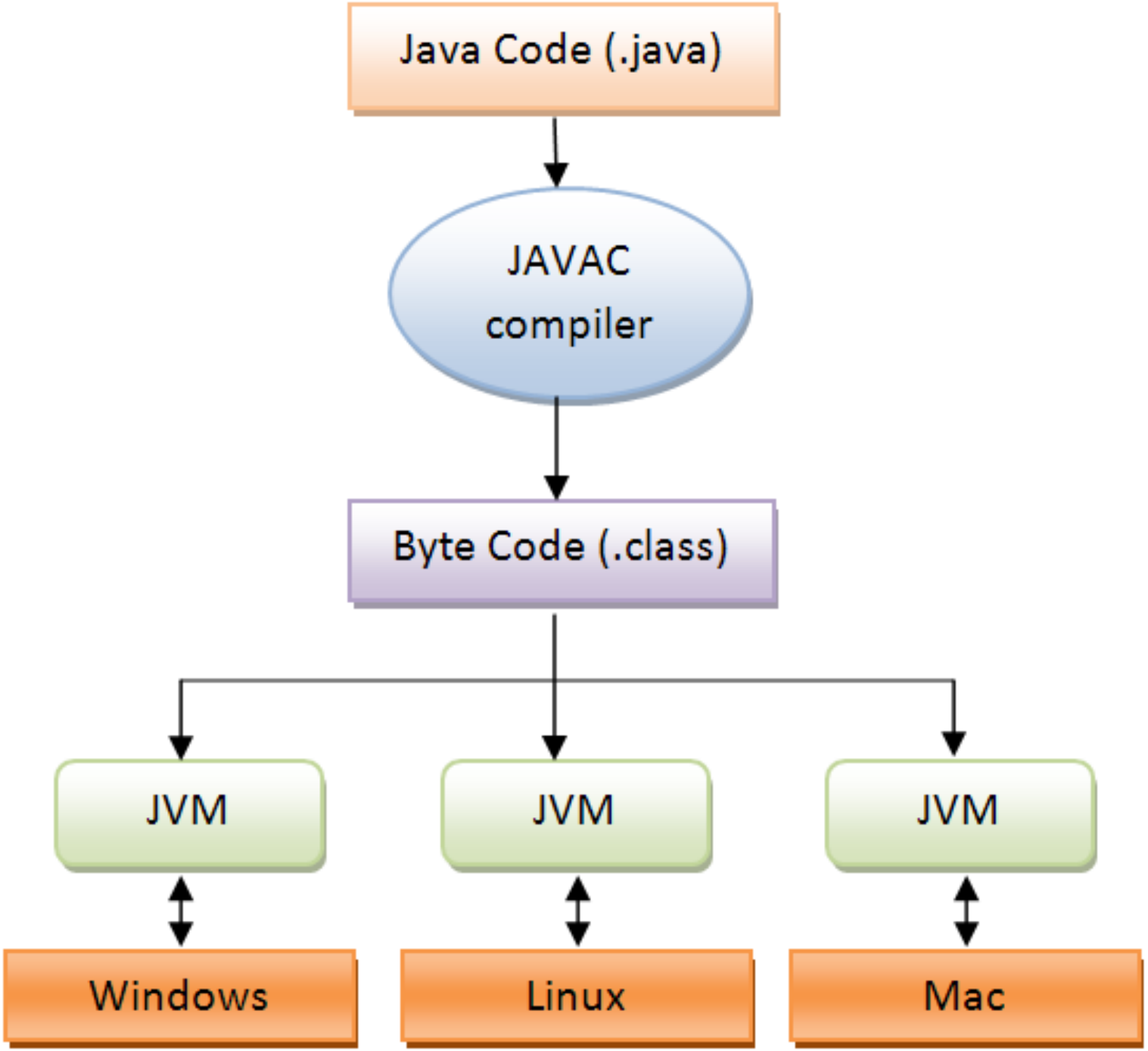
Java

Il linguaggio Java

- Progettato da Sun Microsystems (ora Oracle) nel 1995/96
  - Useremo la Standard Edition (versione 8)
- Esistono anche
  - Java Enterprise Edition
    - Costruita sopra Java SE, offre servlets e JSP, EJB…
    - Per sviluppare applicazioni "server side"
  - Java Micro Edition
    - Per piccoli dispositivi
    - Ormai obsoleta (Andriod, …)

Caratteristiche

- Object-oriented (OO)
- Supporto alla distribuzione
  - RMI e altri strumenti di distribuzione
- Indipendente dalla piattaforma
  - Bytecode e Java Virtual Machine
- Sicuro
  - Esecuzione in una "sandbox" che garantisce che non si possa danneggiare l'host


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

Classi come astrazioni

La struttura di un programma Java

- Un programma Java è organizzato come un insieme di classi
  - Ogni classe corrisponde a una dichiarazione di tipo
- Una classe contiene dichiarazioni di variabili (**attributi**) e di funzioni (**metodi**)
  - Il valore degli attributi caratterizzano le singole istanze (**oggetti**) di una classe
  - I metodi servono per manipolare le istanze
    - Hanno sintassi analoga alle funzioni C
- Il programma principale è rappresentato da un metodo speciale di una classe, detto main

Esempio di classe in Java

```
public class Data {
      private int giorno;
      private int mese;
      private int anno;

// restituisce il giorno
      public int leggiGiorno(){…}

// restituisce il mese
      public int leggiMese(){…}

// restituisce l'anno
      public int leggiAnno(){…}
}
```

Il primo programma Java

```
public class HelloWorld {
	public static void main(String[] args){
		System.out.println("Hello world!");
	}
}
```

Il costrutto class

- Una classe può essere vista come un tipo definito dall'utente che specifica le operazioni utilizzabili sul tipo stesso
- Il tipo può essere usato per dichiarare altre variabili
  - int a, b;
    - Dichiara due variabili a e b sulle quali è possibile fare tutte le operazioni predefinite per il tipo int
  - Data d;
    - Dichiara una variabile d sulla quale è possibile fare tutte le operazioni definite nella classe Data
- Definisce un **tipo di dato astratto**

Oggetti

- Tutti gli oggetti della stessa classe hanno la stessa struttura
  - Il numero e tipo dei loro attributi è lo stesso
  - Ad esempio, la classe Data definisce tutte le date possibili, tramite gli attributi giorno, mese, anno
- Ogni oggetto, in ogni istante dell'esecuzione del programma, è caratterizzato da uno **stato**, che è dato dal valore degli attributi dell'oggetto
  - Lo stato dell'oggetto "oggi" di tipo Data è definito dal valore degli attributi giorno, mese, anno corrispondenti alla data di oggi

Accesso ad attributi e metodi

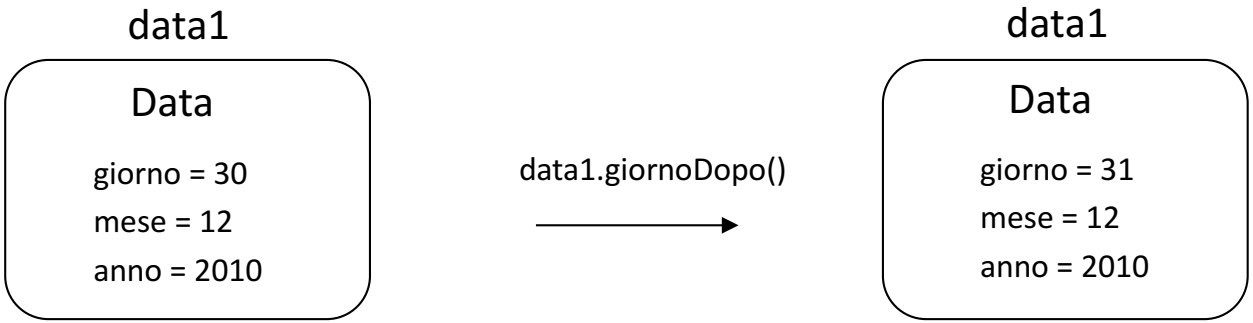
- Tramite la "notazione punto"
- Esempio:

```
Data d;
int x;
//codice che inizializza d (lo vedremo dopo)
x = d.leggiGiorno();
```

  - **Invochiamo il metodo** leggiGiorno() sull'oggetto d: restituisce valore del giorno della data d
  - È come se leggiGiorno() avesse come argomento implicito d: in C scriveremmo: leggiGiorno(d);
  - Si dice anche che all'oggetto d **inviamo il messaggio** leggiGiorno

Cambiamenti di stato

- Lo stato degli oggetti può cambiare nel tempo, attraverso l'invocazione di metodi opportuni



- Esistono alcuni casi in cui gli oggetti sono **immutabili**, cioè non possono essere modificati
  - Ad esempio la classe predefinita String

metodo giornoDopo()

- Per modificare lo stato, il metodo deve potere accedere ai campi dell'oggetto su cui è stato chiamato
- Nella definizione di un metodo ci si può riferire direttamente (senza notazione punto) ad attributi e metodi dell'oggetto sul quale si sta lavorando

```
public void giornoDopo() {
    giorno++;
    if (giorno > 31) {
            giorno = 1;
            mese++;
    }
    if (mese > 12) {
            mese = 1;
            anno++;
    }
}
```

Private e public

- Attraverso i metodi "public" di una classe è possibile vedere qual è lo stato di un oggetto ...

```
Data d;
int x;

x = d.leggiGiorno();
```

- …ma non accedere ai dati "private"

```
if (d.mese == 12) //Errore di compilazione!
```

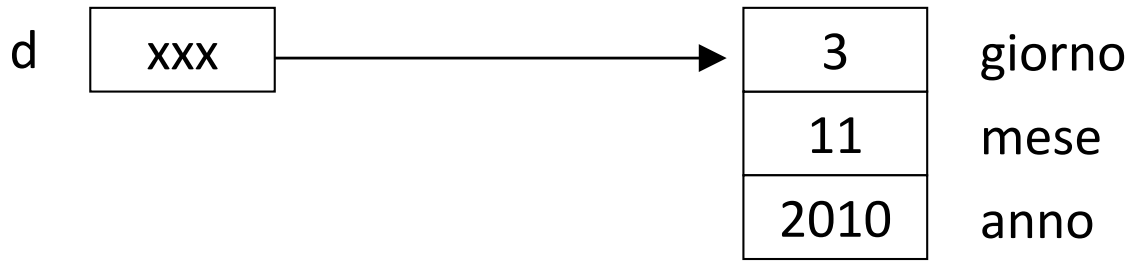
Tipi primitivi e variabili

- Tipi numerici:
  - byte: 8 bit
  - short: 16 bit
  - int: 32 bit
  - long: 64 bit
  - float: 32 bit
  - double: 64 bit
- Altri tipi:
  - boolean: true, false
  - char: 16 bit, Unicode

- Dichiarazione
  - byte unByte;
  - int a, b=3, c;
  - char c= ‘h’, car;
  - boolean trovato=false;

Tipi riferimento

- Le variabili di tipo primitivo (numerico, char, bool) contengono **direttamente** il valore
- Tutte le altre contengono **riferimenti** a valori:
  - Tipi definiti dall'utente
  - Array ed enumerazioni
- Ogni variabile dichiarata da una classe contiene un riferimento a un oggetto: un indirizzo di memoria
  - Il valore effettivo dell'indirizzo non è noto e non interessa
- Un oggetto è memorizzato in un'opportuna area di memoria
  - La **variabile oggetto** d, di tipo Data, contiene l'indirizzo della prima cella dell'oggetto



Variabili e tipi riferimento

- Tipi riferimento e primitivi possono essere utilizzati in maniera (quasi) intercambiabile:
  - Dichiarazioni di variabili e parametri, tipo del valore restituito da un metodo
- Le variabili
  - Consentono di accedere agli oggetti
  - Sono **allocate nello stack** quando esegue il metodo dove sono dichiarate
  - Gli oggetti referenziati dalle variabili sono **allocati sullo heap**
  - Le variabili sono deallocate quando il metodo termina

Dichiarazione e inizializzazione

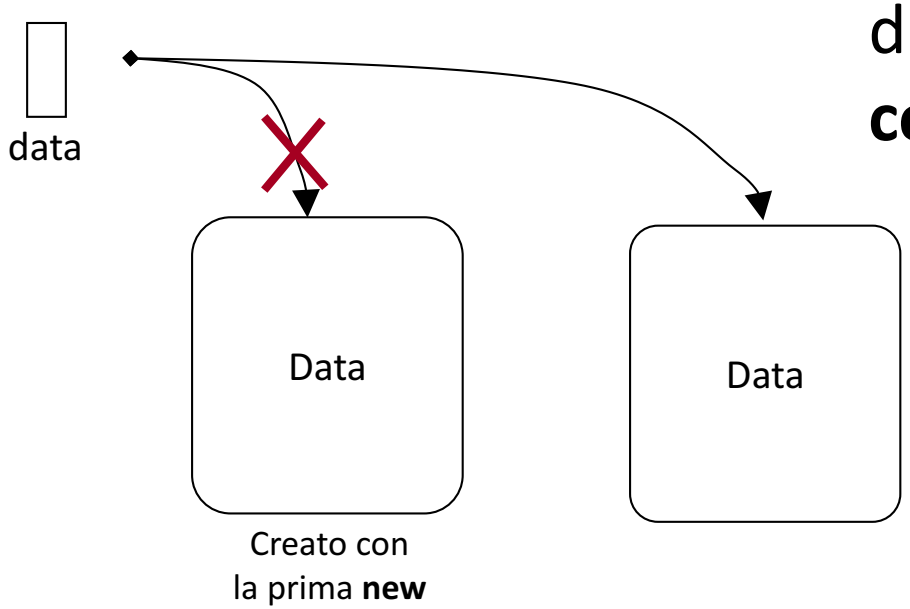
- La dichiarazione di una variabile di tipo riferimento non alloca spazio per un oggetto
  - …ma solo per il riferimento ad un oggetto
- A una variabile di tipo riferimento è assegnato inizialmente il riferimento **null**
  - La variabile non è ancora associata ad alcun oggetto
  - Data d;
    - A questo punto d vale null, perchè non esiste ancora un oggetto di tipo Data
  - Il compilatore riconosce in maniera automatica il possibile uso di variabili non inizializzate

New

- La costruzione di un oggetto si realizza dinamicamente tramite l'operatore **new**
- Esempio:
  - Data d = new Data();
- Effetto di new:
  - "Costruisce" un nuovo oggetto di tipo Data
  - Restituisce il **riferimento** all'oggetto appena creato
    - Il riferimento viene conservato nella variabile d per potere manipolare l'oggetto
  - Data() è un metodo particolare chiamato **costruttore**
    - Ha lo stesso nome della classe, si riconosce come metodo dalle parentesi "()"

Dichiarazione e creazione

```
Data data;
data = new Data();
data = new Data();
```



- Il riferimento al primo oggetto Data è perso
- Non c'è più modo di accedere all'oggetto
- L'oggetto verrà distrutto dal **garbage collector**

Costruttori di default

- Se non si definisce nessun costruttore, il compilatore fornisce il **costruttore di default** (senza parametri), che svolge le seguenti funzioni:
  - Alloca lo spazio per gli attributi di tipo primitivo
  - Alloca lo spazio per i riferimenti agli attributi di tipo definito dall'utente
  - Inizializza a null tutti i riferimenti, a 0 tutte le variabili numeriche, a false tutti i boolean
- Se lo sviluppatore fornisce dei costruttori personalizzati, il compilatore **non** fornisce quello di default

I costruttori: esempio

```
public class C {
    private int i1;
    private int i2;
}

public class Esempio {
    public static void main(String args[]) {
     C x;
     x = new C();
     x = new C(5,7);
    }
}
```

Corretto, non c'è costruttore personalizzato, il compilatore fornisce quello di default

Errore a compile-time, non c'è un costruttore che prende due parametri

Costruttori personalizzati

```
public class C {
    private int i1;
    private int i2;
    public C(int a1, int a2) {
     i1 = a1; i2 = a2;
    }
    public C(int a) {
     i1 = a; i2 = a;
    }
}

public class Esempio {
public static void main(String args[]) {
    C x, y, z;

    x = new C();

    y = new C(1);

    z = new C(1,4);
}
```

Errore a compile-time, avendo fornito costruttori personalizzati, il compilatore non fornisce quello di default

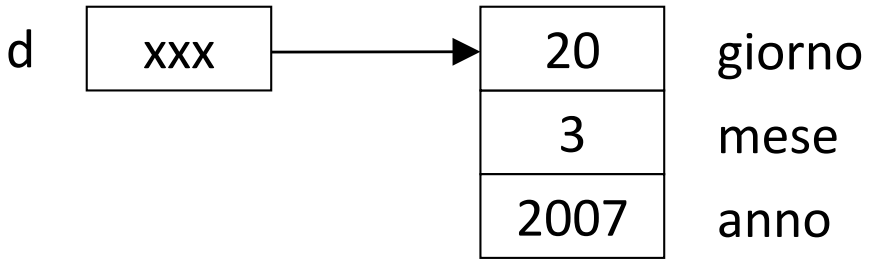
Corretto, usa il secondo costruttore

Corretto, usa il primo costruttore

Esempio di costruttore

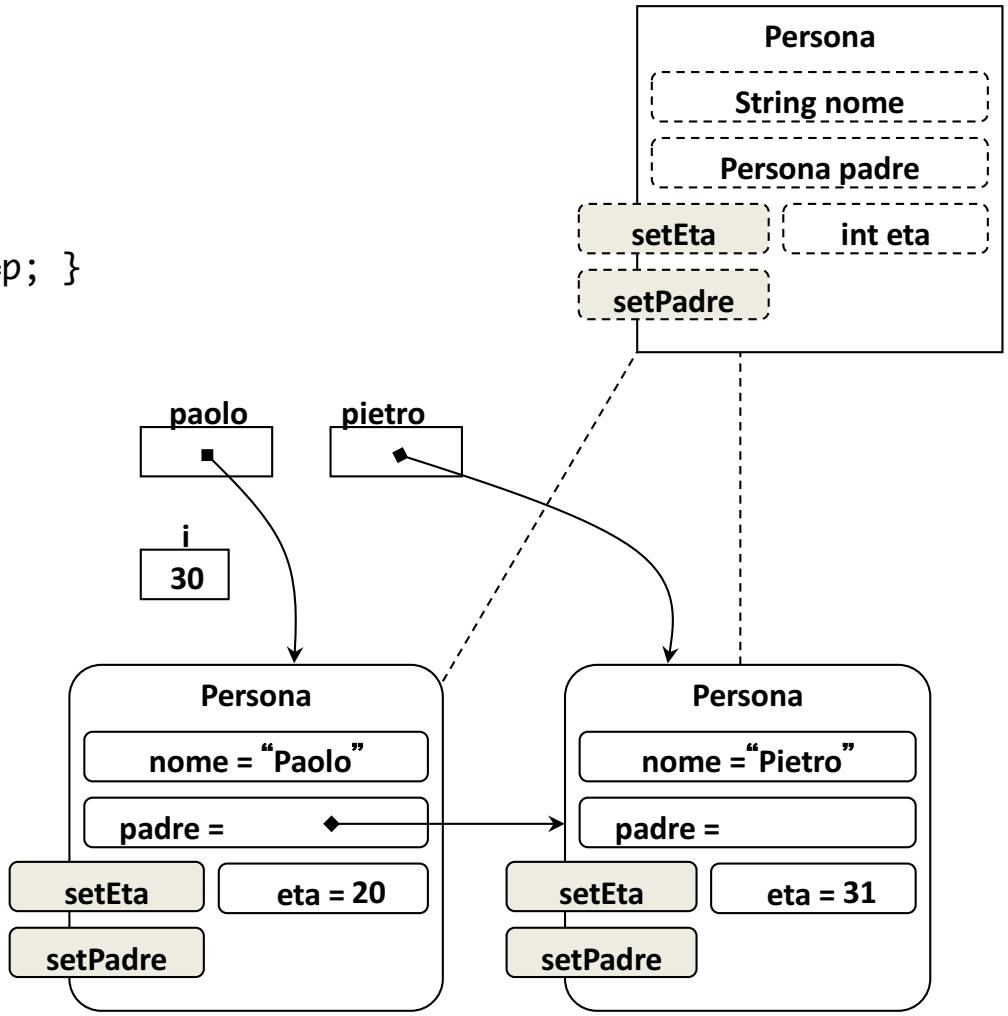
```
public Data(int g, int m, int a){
   giorno = g;
   mese = m;
   anno = a;
}

Data d = new Data(20,3,2007);
```

- Crea un oggetto d di tipo Data e lo inizializza al 20/03/2007
- Un'implementazione più sofisticata potrebbe controllare che la data proposta sia legale, eventualmente rifiutando il 31/2/2007



Inizializzazione e accesso

```
public class Persona {
    private String nome;
    private Persona padre;
    private int eta;
    public Persona(String n) {
      nome = n; padre = null; eta = 0;
    }
    public void setPadre(Persona p) { padre =p; }
    public void setEta(int e) { eta = e; }
}

public class Esempio {
    public static void main(String[] args) {
      Persona paolo, pietro;
      int i = 20;
      paolo = new Persona("Paolo");
      paolo.setEta(i);
      i = 30; // quanto vale paolo.eta?
      pietro = new Persona("Pietro");
      pietro.setEta(i);
      paolo.setPadre(pietro);
      // quanto vale paolo.padre.eta?
      pietro.setEta(i+1);
      // quanto vale paolo.padre.eta ?
    }
}
```



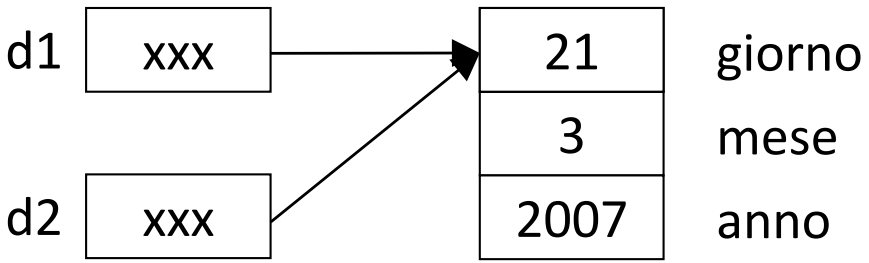
Creazione e distruzione

- Se l'implementazione deve essere private, l'unico modo per inizializzare un oggetto è specificare uno o più costruttori
  - La creazione di un oggetto comporta sempre l'invocazione di un costruttore
- Il costruttore svolge due operazioni fondamentali, obbligandoci a definirle insieme:
  - L'allocazione della memoria necessaria a contenere l'oggetto
  - L'inizializzazione dello spazio allocato, assegnando opportuni valori
- A differenza di altri linguaggi, in Java non è necessario deallocare esplicitamente gli oggetti
  - Il **garbage collector** si occupa di questo: è una routine di sistema che provvede automaticamente a liberare memoria quando serve
  - Non necessariamente disponibile in altri linguaggi che supportano OO, come C++

Condivisione (sharing o aliasing)

- Un oggetto è condiviso tra due variabili se entrambe accedono a esso
- L'assegnamento di variabili di tipo riferimento genera condivisione
- Se oggetti condivisi sono modificabili, le modifiche apportate attraverso una variabile sono visibili anche attraverso l'altra

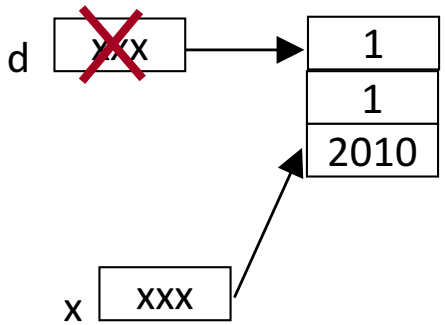
```
Data d1, d2;
d1 = new Data(20,3,2007);

d2 = d1;
d2.giornoDopo();

// Cosa stampa?
System.out.println(d1.getGiorno());
```



Allocazione stack vs. heap

- Quando un metodo termina, tutte le variabili del corrispondente record di attivazione sono distrutte
- …però gli oggetti creati sullo heap **non** sono necessariamente distrutti

```
public Data foo() {
 Data d = new Data(1,1,2005);
 return d;
}
public static void main(String[] args) {
 Data x = foo();
}
```



- Il riferimento d esiste solo sullo stack durante l'esecuzione di foo(), ma viene deallocato quando foo() termina
  - L'oggetto costruito continua a vivere sullo heap !!!
  - Il garbage collector non pulisce la memoria perchè x continua ad avere un riferimento all'oggetto sullo heap

Tipi array

- Dato un tipo T (predefinito o definito dall'utente) un array di T è definito come
  - `T[]`
- Similmente sono dichiarati gli array multidimensionali
  - `T[][] T[][][] …`
- Esempi
  - `int[] float[] Persona[]`

Dichiarazione e inizializzazione

- Dichiarazione
  - `int[] ai1, ai2;`
  - `float[] af1;`
  - `double ad[];`
  - `Persona[][] ap;`
- Inizializzazione
  - `int[] ai={1,2,3};`
  - `double[][] ad={{1.2, 2.5}, {1.0, 1.5}};`

Il caso degli array

- In mancanza di inizializzazione, la dichiarazione di un array non alloca spazio per gli elementi dell'array
- L'allocazione si realizza dinamicamente tramite l'operatore:
  - new <tipo> [<dimensione>]

```
int[] i= new int[10];
i={10,11,12};
float[][] f= new float[10][10];
Persona[] p= new Persona[30];
```

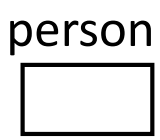
- Se gli elementi non sono di un tipo primitivo, l'operatore "new" alloca **solo lo spazio per i riferimenti**

Array di oggetti: Definizione

A

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

E' definita solo la variabile person, l'array vero e proprio non esiste

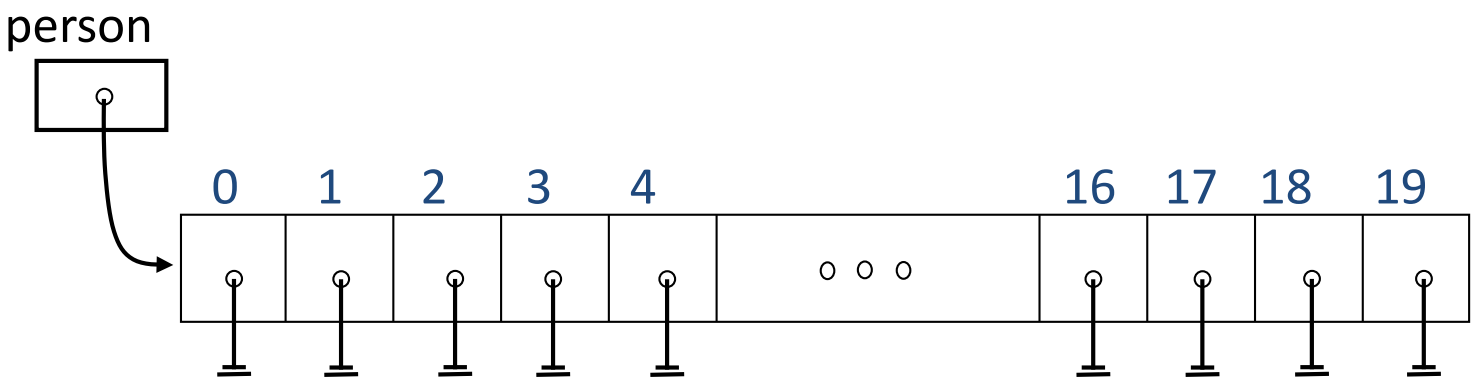


Array di oggetti: Definizione

B

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Ora l'array è stato creato ma i diversi oggetti di tipo Person non esistono ancora…

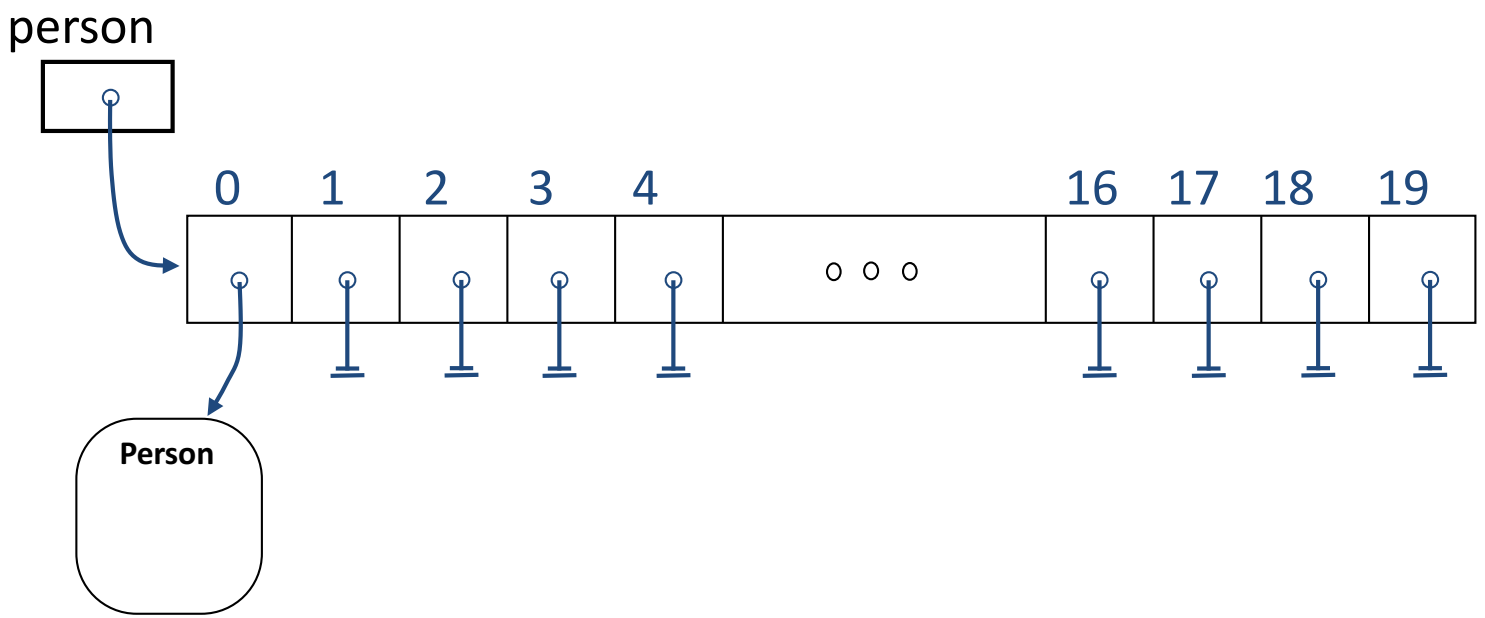


Array di oggetti: Definizione

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

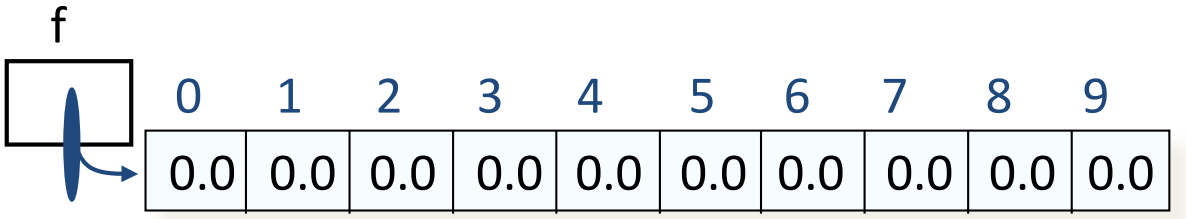
C

Un oggetto di tipo persona è stato creato e un riferimento a tale oggetto è stato inserito in posizione 0



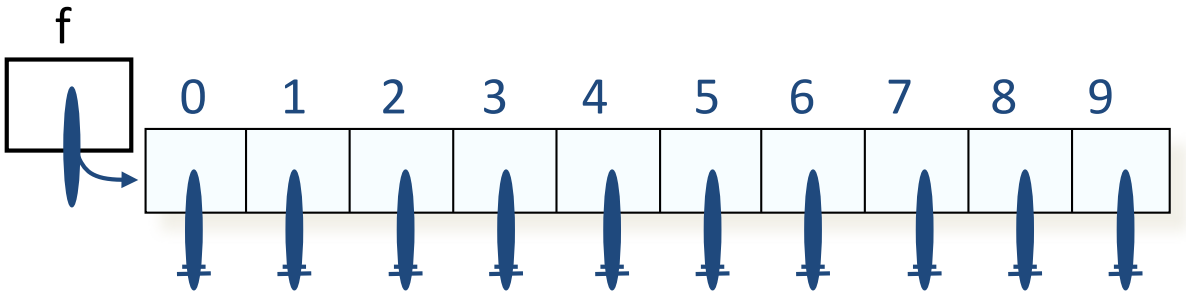
Array di oggetti vs. array di tipi base

L'istruzione: float f[] = new float[10];
crea un oggetto di tipo array di float e alloca spazio per 10 float



L'istruzione: Person p[] = new Person[10];
crea un oggetto di tipo array di Person e alloca spazio per 10 riferimenti
a oggetti di tipo Person

Non viene allocato spazio per gli oggetti veri e propri



Loop generalizzato per collezioni

- Se abbiamo una collezione C di elementi di tipo T, possiamo iterare su di essi scrivendo

```
for (T x: C) { //si legge “for each x in C”
  // esegui azioni su x
}
```

- È equivalente a scrivere

```
for (int i = 0; i < C.size(); i++) {
T x = C.get(i);
// esegui azioni su x
}
```

- Anche gli array sono collezioni, quindi …

Iterare negli array

```
int sum(int[] a) {
   int result = 0;
   for (int  n  : a) //si legge “for each n in a”
    result += n;
   return result;
}
```

- La variabile n indica il **generico elemento** dell'array, non l'indice!
- Un esempio più chiaro: array di Person

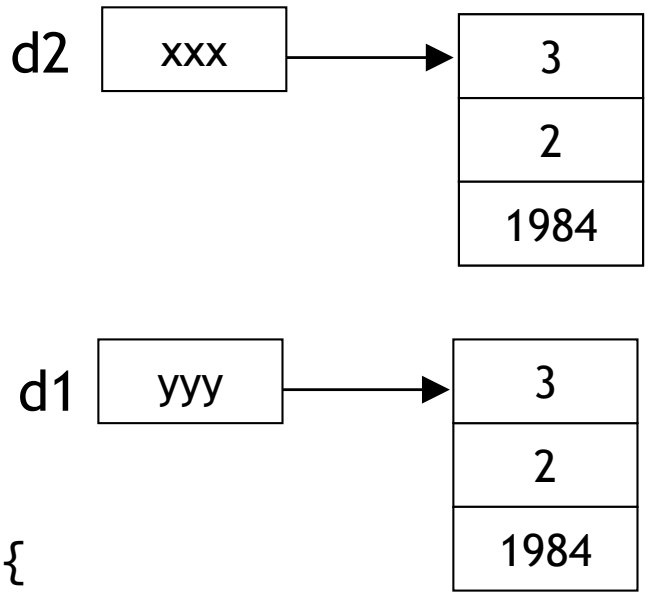
```
int sumAge(Person[] a) {
   int result = 0;
   for (Person  p: a)
    result += p.getAge();
   return result;
}
```

Chiamata di metodi

- Vengono valutati i parametri attuali
- Viene creato il record di attivazione sullo stack
  - Spazio per i parametri formali
  - Spazio per le variabili locali
- Esecuzione del corpo
- Regola per il **passaggio parametri**
  - I parametri il cui tipo è uno dei tipi semplici sono passati per **copia**
  - I parametri il cui tipo è un tipo riferimento sono passati per **riferimento**

Passaggio parametri

```
public class Data {
   private int giorno;
   private int mese;
   private int anno;

   public void copiaIn(Data d){
    d.giorno = giorno;
    d.mese = mese;
    d.anno = anno;
   }

   public static void main(String[] args){
    Data d1 = new Data(3,2,1984);
    Data d2 = new Data(1,1,1990);
    d1.copiaIn(d2);
   }
}
```



Visibilità dei nomi

- Le variabili locali ad un metodo (o i parametri formali) possono mascherare gli attributi della classe
- Soluzione:
  - La pseudo-variabile **this** contiene un riferimento all'oggetto corrente e può essere utilizzata per aggirare eventuali mascheramenti

Aggirare mascheramenti

- La pseudo-variabile this contiene un riferimento all'oggetto corrente e può essere utilizzata per aggirare eventuali mascheramenti

```
public class Automobile {
    private String colore, marca, modello;

    public void trasforma(String marca, String modello){
            this.marca=marca;
            this.modello=modello;
    }
    …
    public static void main(String[] args){
            Automobile a = new Automobile();
            a.trasforma("Ford", "T4");
    }
}
```

- Quindi a.marca diventa "Ford", a.modello diventa "T4"

Un altro esempio

```
public class Data {
    private int giorno;
    private int mese;
    private int anno;

    public void giornoDopo(){
     this.giorno++;
     if (this.giorno > 31){
         this.giorno = 1;
         this.mese++;
     }
     …
    }

    public static void main(String[] args){
     Data d1, d2; //d1 e d2 inizializzate qui
     d1.giornoDopo(); //In giornoDopo this è lo stesso riferimento di d1
     d2.giornoDopo(); //In giornoDopo this è lo stesso riferimento di d2
    }
}
```

Restituire un riferimento

- La pseudo-variabile this può essere utilizzata per restituire un riferimento all'oggetto corrente

```
public class InsiemeDiInteri {
    …
    public InsiemeDiInteri inserisci(int i){
            //modifica this inserendovi l'elemento i
            return this; //restituisce l'insieme modificato
    }
…
}

InsiemeDiInteri x,y,z;
//qui x e y sono inizializzati opportunamente
z = (x.inserisci(2)).unione(y)

//utile che inserisci restituisca un insieme
```