Java: Oggetti Riferimenti e Costruttori

Damiano Macedonio

Dipartimento di Informatica, Università degli Studi di Verona

Corso di Programmazione per Bioformatica

lezione del 28 marzo 2014

Classe **Specie**

Specie
- nome: String - popolazione: int - tassoCrescita: double
+ setSpecie(nome: String, popolazione: int, crescita: double): void + toString(): String

Implementazione

```
1  public class Specie {
2    private String nome;
3    private int popolazione;
4    private double crescita;
5
6    public void setSpecie(String nome, int popolazione, double crescita) {
7      this.nome = nome;
8      this.popolazione = popolazione;
9      this.crescita = crescita;
10   }
11
12   public String toString() {
13     return nome + " " + popolazione + " " + crescita;
14   }
15 }
```

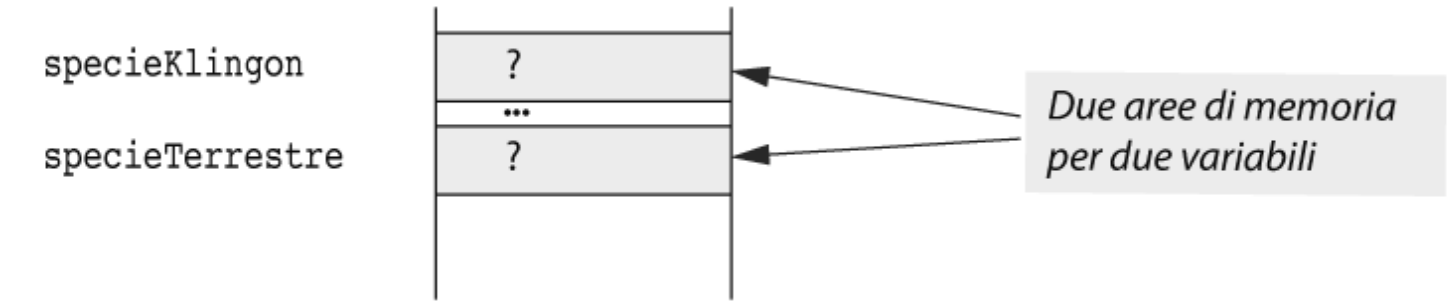
Le variabili di tipo classe

Ciascuna variabile è implementata come un'area di memoria.

- Se la variabile è di tipo primitivo, il suo valore è immagazzinato nell'area di memoria assegnata alla variabile
- Se la variabile è di tipo classe, essa contiene l'indirizzo di memoria dell'oggetto a cui fa riferimento la variabile.
- L'oggetto stesso non è memorizzato nella variabile, ma in un'altra area di memoria.
- L'indirizzo di questa memoria è detto riferimento all'oggetto (reference).

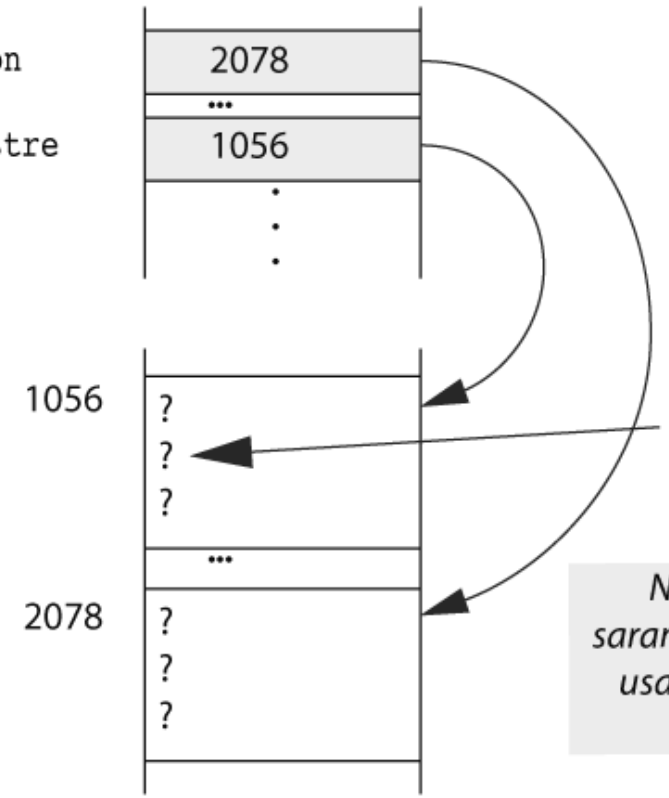
Definizione di variabili

```
SpecieQuartaProva specieKlingon, specieTerrestre;
```



Creazione e assegnamento

```
specieKlingon = new SpecieQuartaProva();
specieTerrestre= new SpecieQuartaProva();
```

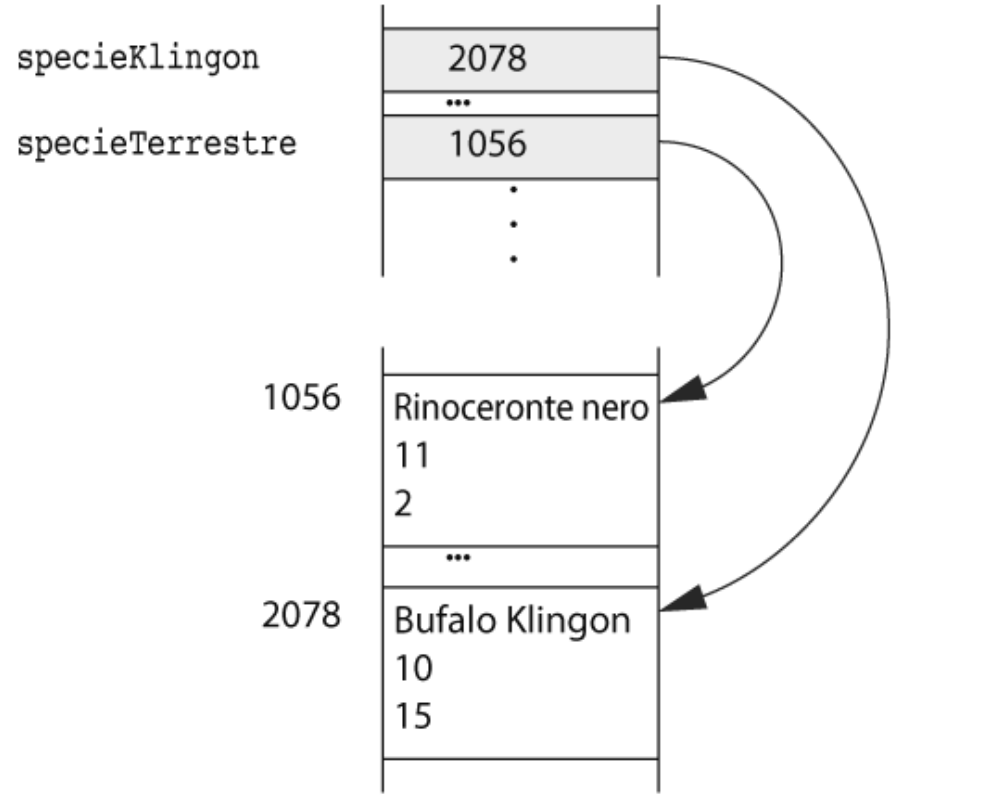


In realtà, come si vedrà nel prossimo capitolo, i punti interrogativi sono dei valori ben precisi.

Non si sa che indirizzi di memoria saranno usati. In questa figura sono stati usati 1056 e 2078, ma possono essere indirizzi qualsiasi.

Scrittura sui campi

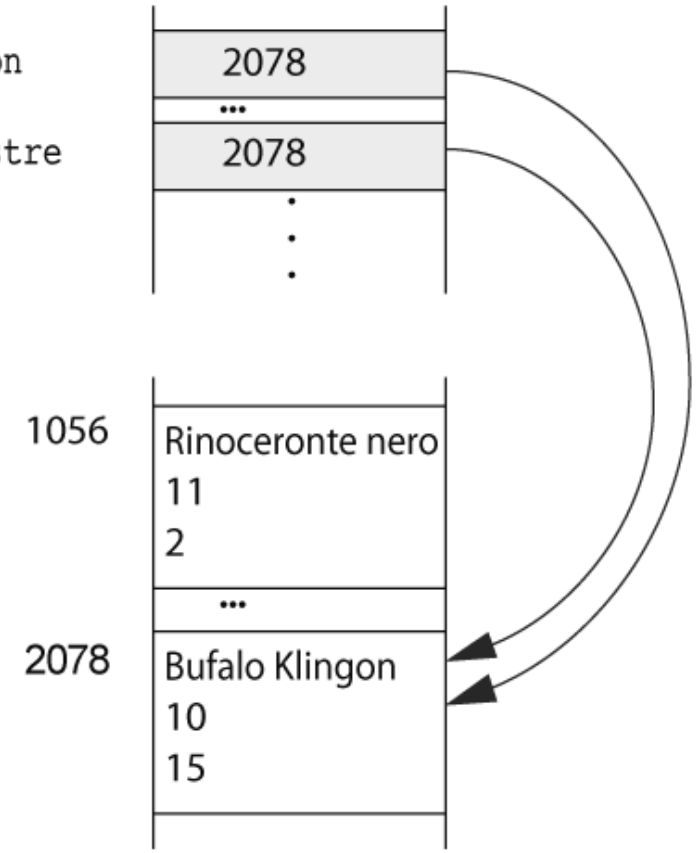
```
specieKlingon.setSpecie("Bufalo Klingon", 10, 15);
specieTerrestre setSpecie("Rinoceronte nero", 11, 2);
```



Assegnamento tra variabili

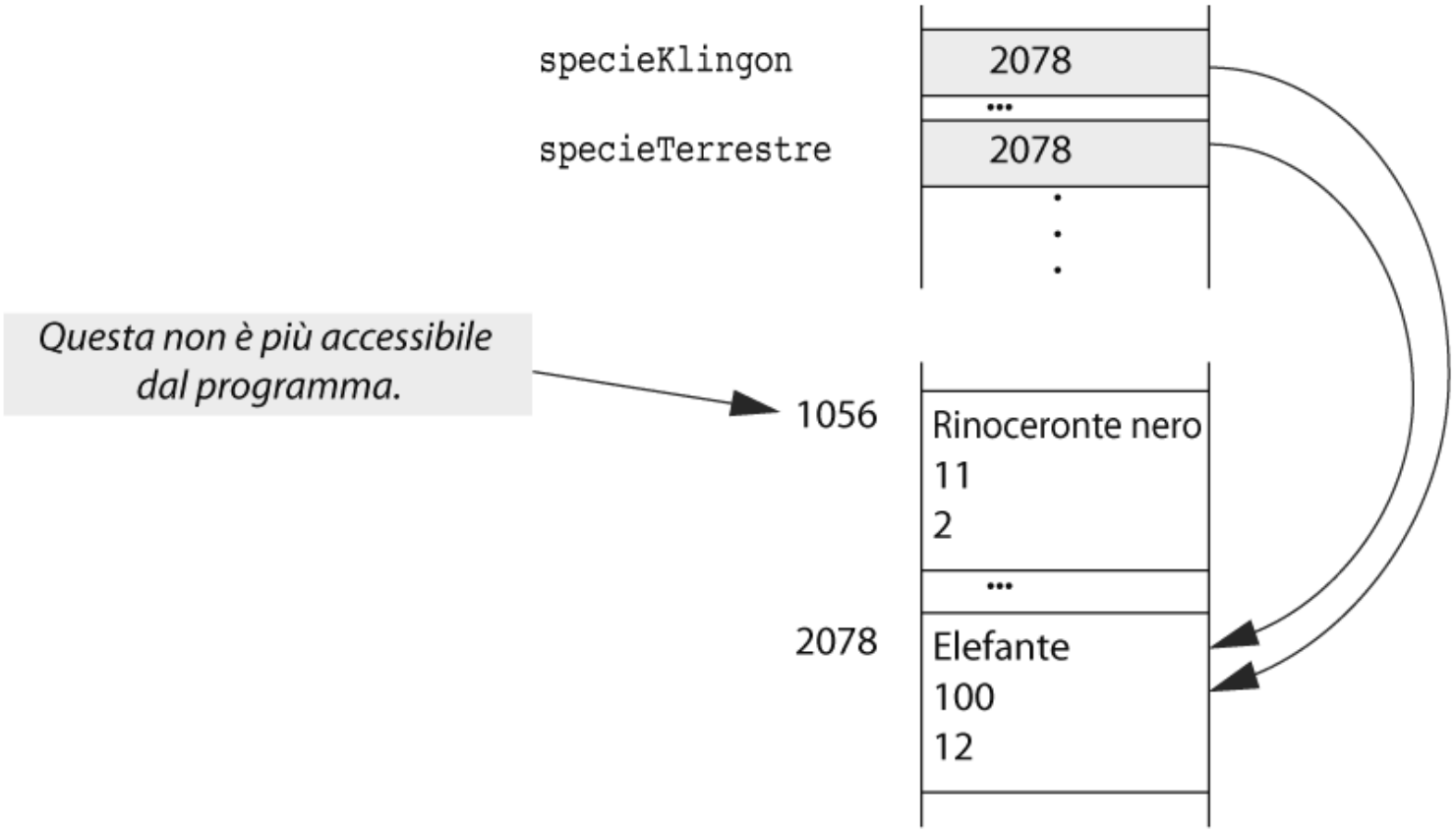
```
specieTerrestre = specieKlingon;
```

`specieKlingon` e `specieTerrestre` puntano allo stesso oggetto (sono due nomi dello stesso oggetto)



Aliasing: due nomi per lo stesso oggetto

```
specieTerrestre.setSpecie ("Elefante", 100, 12);
```



Passando al codice...

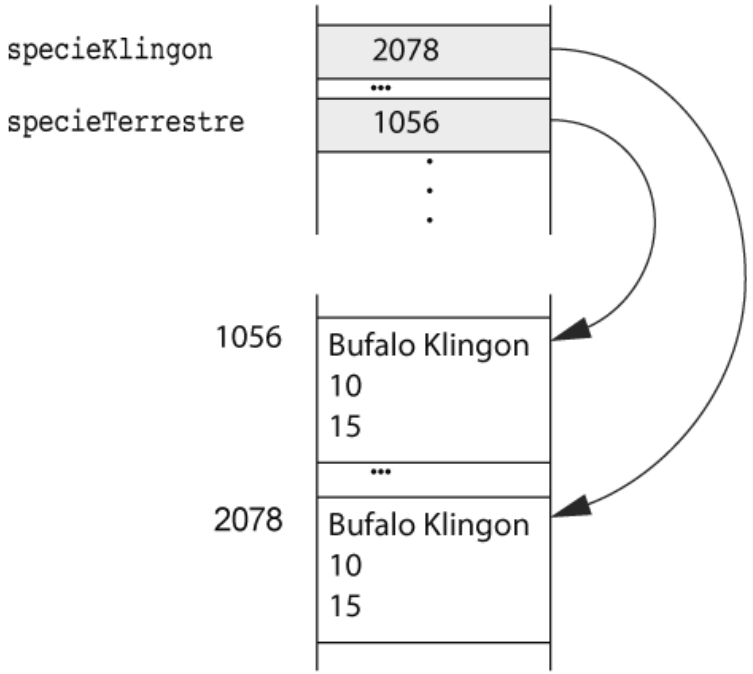
```
1 specieTerrestre = specieKlingon;
2 specieTerrestre.setSpecie("Elefante", 100, 12);
3 System.out.println("specie Terrestre:");
4 System.out.print(specieTerrestre.toString());
5 System.out.println("specie Klingon:");
6 System.out.print(specieKlingon.toString());
```

Output

```
specie Terrestre: Elefante 100 12.0
specie Klingon: Elefante 100 12.0
```

Operatore == tra oggetti

```
specieKlingon.setSpecie("Bufalo Klingon", 10, 15);
specieTerrestre.setSpecie("Bufalo Klingon", 10, 15);
```




```
if (specieKlingon == specieTerrestre)
    System.out.println("Sono UGUALI");
else
    System.out.println("NON sono uguali");
```

L'output è `NON sono uguali`, *poiché* `2078` *non è uguale a* `1056`.

Il metodo `equals()`

Se due oggetti contengono gli stessi dati (ovvero le loro variabili di istanza contengono gli stessi valori), ma hanno diversi indirizzi di memoria, essi risultano diversi secondo l'operatore ==

L'operatore == controlla solo se gli indirizzi di memoria sono gli stessi

Ogni volta che si definisce una classe si dovrebbe definire un metodo `equals` che verifica se gli oggetti sono uguali, nel senso che si trovano nello stesso stato: le loro variabili di istanza hanno gli stessi valori.

Come implementare il metodo `equals`

Per la classe **Specie**: due oggetti sono uguali se rappresentano specie con lo stesso nome, la stessa popolazione e lo steso tasso di crescita

```
public boolean equals(Specie other) {
  return (nome.equalsIgnoreCase(other.nome)
      && (popolazione == other.popolazione)
      && (crescita == other.crescita));
}
```

Non esiste una definizione unica di `equals`, ma dipende da come si intende usare la classe.

Si deve sempre usare `equals` per identificare il nome del metodo che verifica se due oggetti sono uguali.

Se non si definisce il metodo `equals` per una classe, Java ne crea uno automaticamente, secondo una definizione di default. Tuttavia questo potrebbe non comportarsi come si vorrebbe.

Metodi Booleani

Per la classe **Specie**:

```
public boolean isPopolazioneMaggiore(Specie other) {
  return popolazione > other.popolazione;
}

public boolean isEstinta() {
  return popolazione == 0;
}
```

⋮

```
if (specieKlingon.isEstinta())
  System.out.println("non ci son piu' esemplari della specie")
else
  System.out.println("possiamo ancora incontrare degli esemplari")
```

Per convenzione, si usa il termine `is` come prefisso nei metodi booleani per chiarirne lo scopo.

Parametri di tipo classe

Un parametro formale di tipo classe è una variabile locale che contiene l'indirizzo di memoria di un oggetto del tipo classe specificato.
Quando viene invocato il metodo, il parametro viene inizializzato con l'indirizzo di memoria dell'argomento passato all'invocazione del metodo.

Aliasing

Il parametro formale è un nome alternativo per l'oggetto fornito come argomento in un'invocazione di metodo.

Qualsiasi azione intrapresa con un parametro formale di tipo classe, viene intrapresa anche con l'argomento passato nell'invocazione di metodo. L'oggetto passato può venire modificato!

Esempio - 1

```
public static void m(Specie s1, Specie s2, int n){
  s1.setSpecie("Bisonte muschiato", 100, n);
  n = 24;
  s2.setSpecie("Rana toro", 100, n);
}

public static void main(String[] args){
  Specie primaSpecie = new Specie();
  Specie secondaSpecie = new Specie();
  int n = 12;

  primaSpecie.setSpecie("Foca monaca", 50, 2);
  secondaSpecie.setSpecie("Paguro bernardo", 25, 4);

  m(primaSpecie, secondaSpecie, n);

  System.out.print("primaSpecie: ");
  System.out.println(primaSpecie.toString());
  System.out.print("secondaSpecie: ");
  System.out.println(secondaSpecie.toString());
  System.out.print("n: ");
  System.out.println(n);
}
```

Output - 1

```
public static void m(Specie s1, Specie s2, int n){
  s1.setSpecie("Bisonte muschiato", 100, n);
  n = 24;
  s2.setSpecie("Rana toro", 100, n);
}
 ...
  primaSpecie.setSpecie("Foca monaca", 50, 2);
  secondaSpecie.setSpecie("Paguro bernardo", 25, 4);

  m(primaSpecie, secondaSpecie, n);

  System.out.print("primaSpecie: ");
  System.out.println(primaSpecie.toString());
  System.out.print("secondaSpecie: ");
  System.out.println(secondaSpecie.toString());
  System.out.print("n: ");
  System.out.println(n);
...
```

```
primaSpecie: Bisonte muschiato 100 12.0
secondaSpecie: Rana toro 100 24.0
n: 12
```

Esempio - 2

```
public static void m(Specie s1, Specie s2, int n){
  s1.setSpecie("Bisonte muschiato", 100, n);
  n = 24;
  s2.setSpecie("Rana toro", 100, n);
}

public static void main(String[] args){
  Specie primaSpecie = new Specie();
  Specie secondaSpecie = primaSpecie; // assegnamento
  int n = 12;

  primaSpecie.setSpecie("Foca monaca", 50, 2);
  secondaSpecie.setSpecie("Paguro bernardo", 25, 4);

  m(primaSpecie, secondaSpecie, n);

  System.out.print("primaSpecie: ");
  System.out.println(primaSpecie.toString());
  System.out.print("secondaSpecie: ");
  System.out.println(secondaSpecie.toString());
  System.out.print("n: ");
  System.out.println(n);
}
```

Output - 2

```
public static void m(Specie s1, Specie s2, int n){
  s1.setSpecie("Bisonte muschiato", 100, n);
  n = 24;
  s2.setSpecie("Rana toro", 100, n);
}
 ...
  primaSpecie.setSpecie("Foca monaca", 50, 2);
  secondaSpecie.setSpecie("Paguro bernardo", 25, 4);

  m(primaSpecie, secondaSpecie, n);

  System.out.print("primaSpecie: ");
  System.out.println(primaSpecie.toString());
  System.out.print("secondaSpecie: ");
  System.out.println(secondaSpecie.toString());
  System.out.print("n: ");
  System.out.println(n);
...
```

```
primaSpecie: Rana toro 100 24.0
secondaSpecie: Rana toro 100 24.0
n: 12
```

Esempio - 3

```
public static void m(Specie s1){
  Specie secondaSpecie = new Specie();
  secondaSpecie.setSpecie("Rana toro", 100, 24);

  s1 = secondaSpecie; // cambia l'oggetto associato a s1
  s1.setSpecie("Bisonte muschiato", 100, 12);

}

public static void main(String[] args){
  Specie primaSpecie = new Specie();

  primaSpecie.setSpecie("Foca monaca", 50, 2);

  m(primaSpecie);

  System.out.print("primaSpecie: ");
  System.out.println(primaSpecie.toString());
}
```

Output

```
primaSpecie: Foca monaca 50 2.0
```

Definire i costruttori

Un costruttore è un particolare metodo che viene invocato quando si utilizza l'operatore `new` per creare un oggetto.

```
Specie specieTerrestre = new Specie();
```

Il costruttore `Specie()` è il costruttore di default fornito automaticamente e da alle variabili di istanza un valore iniziale di default. Tali valori potrebbero non essere quelli desiderati.

La definizione di un costruttore permette di inizializzare le variabili di istanza con specifici valori nel momento in cui l'oggetto viene creato.

Un costruttore può eseguire qualsiasi azione inserita nella sua definizione. Ha essenzialmente lo stesso compito di un metodo `set`, ma crea un oggetto oltre che a inizializzarlo.

Caratteristiche di un costruttore

Un costruttore più avere parametri.

Un costruttore ha lo stesso nome della sua classe.

I costruttori possono avere più definizioni

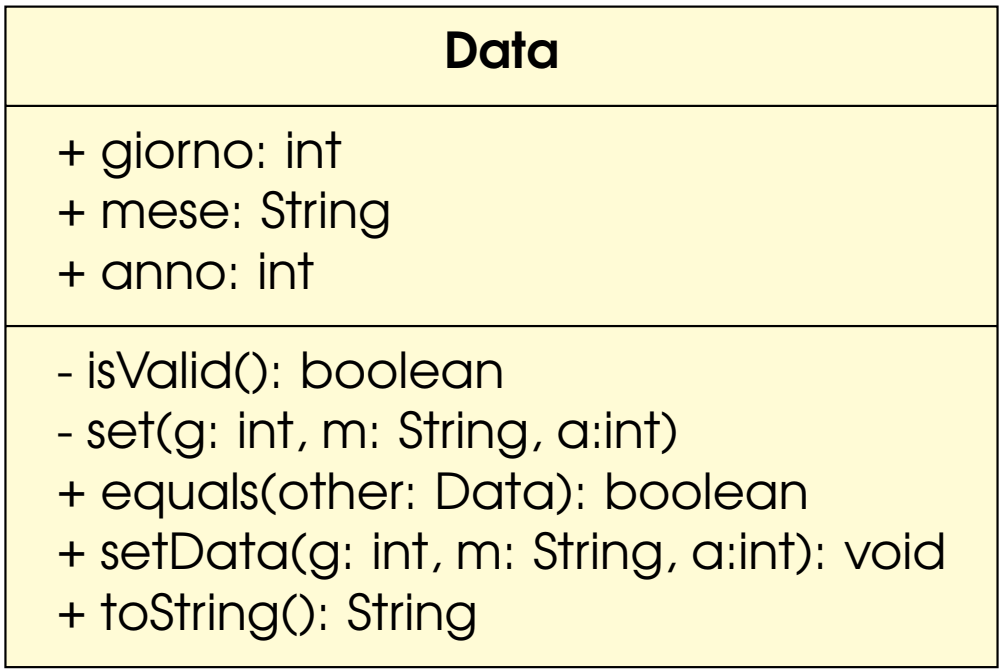
Quando si definisce un costruttore, non si specifica alcun tipo di ritorno e nemmeno `void`.

Se in una classe viene definito almeno un costruttore, non viene aggiunto automaticamente nessun altro contruttore. Anche nel caso in cui venga definito un solo costruttore con parametri.

Quando si crea un oggetto utilizzando l'operatore `new` occorre sempre includere l'invocazione ad uno dei costruttori definiti nella classe, indicando la lista dei parametri.

Una volta creato l'oggetto, l'unico modo per modificarne i valori contenuti è con un metodo `set`.

La classe **Data**



Implementazione: costruttori (prima versione)

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

  if (!isValid())
    System.out.println("Data inesistente");
}
```

Implementazione: costruttori (prima versione)

```
public Data(int g, String m) {
  anno = 2014;
  giorno = g;
  mese = m;

  if (!isValid())
    System.out.println("Data inesistente");
}
```

Implementazione: costruttori (prima versione)

```
public Data(int g, int m, int a) {
  String mesi[] = { "gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno",
      "luglio", "agosto", "settembre", "ottobre", "novembre", "dicembre" };

  giorno = g;
  mese = mesi[m - 1];
  anno = a;

  if (!isValid())
    System.out.println("Data inesistente");
}
```

Creare oggetti

```
Data ultimoNatale = new Data(25, "dicembre", 2013);
Data oggi = new Data(28, "marzo");
Data scoperta = new Data(12, 10, 1492);
Data farlocca = new Data() // chiamata ad un costruttore inesistente!!!
```

Invocare un costruttore da un altro costruttore

Basta utilizzare la parola chiave `this` come se fosse il nome di un metodo in un'invocazione.

L'invocazione deve essere la prima azione eseguita all'interno del corpo del costruttore.

Scrivere Costruttori Interdipendenti

Quando in una classe si scrivono più costruttori, è buona norma identificare quello che gli altri possono invocare con la parola chiave `this`. Scrivendo i costruttori in questo modo, si localizza l'inizializzazione in un solo posto, rendendo le classi meno complesse e meno soggette ad errori.

Implementazione: costruttori (seconda versione)

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

  if (!isValid())
    System.out.println("Data inesistente");
}


public Data(int g, String m) {
  this(g, m, 2014);
}
```

Implementazione: costruttori (seconda versione)

```
public Data(int g, int m, int a) {
  String mesi[] = { "gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno",
      "luglio", "agosto", "settembre", "ottobre", "novembre", "dicembre" };
  this(g, mesi[i -1], a);
}
```

Questa implementazione è sbagliata! Il costruttore deve essere la prima invocazione all'interno del corpo del costruttore!

Implementazione: costruttori (seconda versione)

```
public Data(int g, int m, int a) {
  this(g, mesi[i -1], a);
  String mesi[] = { "gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno",
      "luglio", "agosto", "settembre", "ottobre", "novembre", "dicembre" };
}
```

Questa implementazione è sbagliata! Alla prima riga, l'array `mesi` non è conosciuto!

Con le nostre conoscenze, non possiamo modificare questo costruttore. Serve qualcosa di `static`...

Chiamate a metodi

- Un costruttore può invocare metodi definiti all'interno della sua classe.
- Attenzione però a quando i costruttori invocano metodi pubblici. Grazie alla ereditarietà un'altra classe potrebbe alterare il comportamento dei metodi pubblici e, di conseguenza, alterare il comportamento dei costruttori.
- Per ora, meglio se definiamo `private` ogni metodo chiamato da un costruttore.

Metodi set

```
// visibile all'esterno
public void setData(int g, String m, int a) {
  set(g, m, a);
}

// ad uso interno
private void set(int g, String m, int a) {
  int vecchioGiorno = giorno;
  String vecchioMese = mese;
  int vecchioAnno = anno;

  giorno = g;
  mese = m;
  anno = a;

  if (!isValid()) {
    System.out.println("data risultante inesistente");
    giorno = vecchioGiorno;
    mese = vecchioMese;
    anno = vecchioAnno;
  }
}
```

Costruttori con metodi set

```
public Data(int g, String m, int a) {
  set(g, m, a); // chiama il metodo privato
}


public Data(int g, String m) {
  this(g, m, 2014)
}
```

Se la data non è valida, l'oggetto è inizializzato ai valori di default!

Metodi Booleani

```
1   private boolean isValid() {
2     String mesi[] = { "gennaio", "febbraio", "marzo", "aprile", "maggio", "giugno",
3         "luglio", "agosto", "settembre", "ottobre", "novembre", "dicembre" };
4
5     int giorniPerMese[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
6
7     int numMese;
8     for (numMese = 1; numMese <= 12; numMese++)
9       if (mese.equals(mesi[numMese - 1]))
10        // trovato il mese, ragioniamo sul giorno
11        return giorno >= 1 && giorno <= giorniPerMese[numMese - 1];
12
13    return false;
14  }
15
16  public boolean equals(Data altra) {
17    return giorno == altra.giorno &&
18      anno == altra.anno &&
19      mese.equals(altra.mese);
20  }
```

Metodo `toString`

```
1  public String toString() {
2    String temp = giorno + " " + mese;
3
4    return (temp + " " + this.anno);
5  }
```