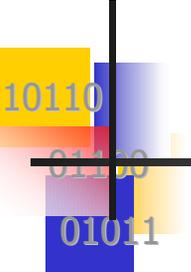


10110

01100

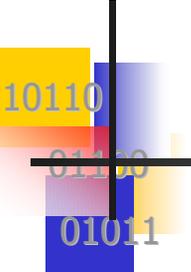
01011

Proprietà delle Classi e degli Oggetti in Java



Proprietà object-oriented di Java

- Definendo le caratteristiche e le operazioni di una **classe** si definiscono implicitamente comportamenti e operazioni degli oggetti che appartengono alla classe.
- Quindi **una classe Java è composta da un insieme di oggetti ed un insieme di metodi associati.**
- **Una classe può essere composta da più sottoclassi**
 - Ad esempio, la **classe dei triangoli** è composta dalle sottoclassi dei triangoli equilateri, dei triangoli isosceli e dei triangoli scaleni.
 - La **classe dei negozi** è composta dalle sottoclassi dei negozi alimentari, di abbigliamento, di elettrodomestici, di automobili ecc.
 - La **classe degli studenti** dell'UNICAL è composta dalle sottoclassi degli studenti di ingegneria, di economia, di scienze, di farmacia, e di lettere.



Proprietà object-oriented di Java

- **Un oggetto è una istanza di una classe.** Per effettuare operazioni su una classe è necessario **creare un oggetto** che è costituito da un insieme di **campi** o **variabili d'istanza**, a cui corrispondono locazioni di memoria, corrispondenti a quelli dichiarati nella classe.
- **Un metodo realizza un insieme di operazioni** sugli oggetti di una classe.
- Gli oggetti di una classe possono essere manipolati **solo tramite i metodi della classe** stessa. I metodi possono essere definiti con i modificatori **public** per indicare che sono visibili anche a metodi non dichiarati nella classe o **private** per indicare che sono visibili solo all'interno della classe. **C'è una terza possibilità che vedremo più avanti.**

Proprietà object-oriented di Java

- **Dichiarazione di oggetti:**

nomeclasse nomeoggetto;

- Esempio:

```
matrice M1, M2;
```

La dichiarazione dell'oggetto presuppone che la classe sia stata già definita.

- **Creazioni di oggetti :**

nomeoggetto = new nomeclasse();

- Esempio:

```
M1 = new matrice();
```

- Nella creazione tra le parentesi () possono essere inseriti valori di inizializzazione per le variabili istanze dell'oggetto. In questo caso occorre avere un **metodo costruttore** che viene eseguito nell'atto della creazione dell'oggetto.

Esempio : la classe tempo

```
class tempo {
    private int ore;           // i campi sono private, tranne separatore
    private int minuti;
    private int secondi;
    public char separatore;   // questi sono i 4 campi dell'oggetto della classe tempo

    public int assegnaTempo(int ora, int min, int sec)
    {
        if (ora >= 0 && ora < 24 && min >= 0 && min < 60
            && sec >=0 && sec <60)
        { this.ore = ora;
          this.minuti = min; // in questo metodo si è usato this ma può essere usato
          this.secondi = sec; // anche negli altri metodi
        }
        return 0;
    }
    else return -1;
}

    public int leggiOra()
{ ore = Console.readInt("Inserire ora: ");
  return ore ;
}
. . . . .
```

Continua →

Esempio : la classe tempo

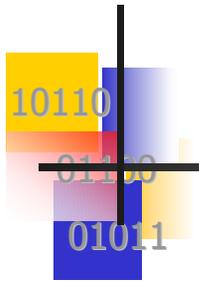
```
. . . . .
public int leggiMinuti()
    {minuti = Console.readInt("Inserire minuti: ");
    return minuti;}

public int leggiSecondi()
    { secondi = Console.readInt("Inserire secondi: ");
    return secondi;}

public void aggiungiOre(int numOre)
    { ore = ore + numOre;
    while (ore > 23)
        ore = ore - 24;
    }
public void visualizza (boolean acapo)
{
    System.out.print(ore);
    System.out.print(separatore);
    System.out.print(minuti);
    System.out.print(separatore);
    if (acapo)
        System.out.println(secondi);
    else
        System.out.print(secondi);
}
}
```

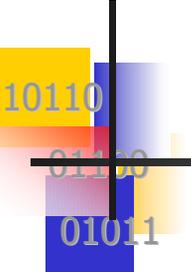
Esempio

```
class Programmtempo
{
    public static void main (String args[])
    { int o, m, s, r;
      tempo miotempo;           // ← object-id
      miotempo = new tempo();   // creazione dell'oggetto
      miotempo.separatore = ':'; // perché è public
      r=miotempo.assegnaTempo(11,30,15);
      miotempo.visualizza(true);
      o=miotempo.leggiOra();
      m=miotempo.leggiMinuti();
      s=miotempo.leggiSecondi();
      miotempo.visualizza(true);
      miotempo.aggiungiOre(6);
      miotempo.visualizza(true);
    }
}
```



Esempio

- Se si dichiarasse un metodo di tipo `private` dentro la classe `tempo`, questo non potrebbe essere utilizzato dalle altre classi.
- Il nome dell'oggetto non è usato nei metodi della classe perché esso è implicito. Usando `this` si può fare riferimento ad esso (vedi esempio).
- Modifiche possibili:
 - Sostituire `assegnaTempo` con il metodo costruttore.
 - Controllare i valori letti nei metodi `leggiOra`, `leggiMinuti` e `leggiSecondi`.
 - Inserire un ciclo nel metodo `main` per leggere e visualizzare orari fino a che non viene letto un valore di particolare.
 - Modificare il metodo `main` in modo che dato un orario vengano visualizzati gli orari corrispondenti in diverse città con differenti fusi orari.
 -



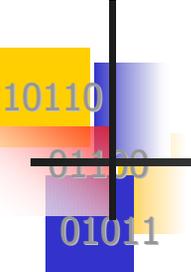
Proprietà delle classi e degli oggetti Java

Analizziamo brevemente le quattro proprietà principali dei linguaggi object-oriented:

- - **Incapsulamento**
- - **Ereditarietà**
- - **Polimorfismo**
- - **Overriding**

Incapsulamento:

- **La proprietà di rendere invisibili i dati e di gestirli solo tramite metodi.**
- In questo modo diversi programmi possono avere diverse visibilità dei dati e si costruiscono delle astrazioni funzionali.
- Questo accade nella vita reale quando usiamo oggetti e macchine senza conoscere il loro contenuto ed il loro funzionamento interno.



Proprietà delle classi e degli oggetti Java

Ereditarietà :

- **Le proprietà di una classe possono essere *ereditate* in tutto o in parte dalle sue sottoclassi che possono avere anche altre specifiche proprietà.**
- La classe genitrice è detta ***superclasse***, mentre la sottoclasse è detta ***classe derivata***. Questo viene realizzato tramite la parola chiave **`extends`**.
- Un oggetto della sottoclasse incapsula tutti i dati della classe genitrice più i suoi dati e può usare tutti i metodi della superclasse risparmiando nella scrittura del codice.
- I metodi e i dati della superclasse non c'è bisogno di riscriverli.

Proprietà delle classi e degli oggetti Java

Ereditarietà

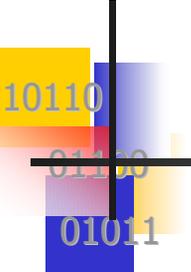
- Se ad esempio, vogliamo estendere la classe `tempo` con una classe `tempo2` che contiene anche i centesimi di secondo si può scrivere:

```
class tempo2 extends tempo
{
    private int centesimi;

    public void assegnaCentesimi (int cent)
    {
        this.centesimi = cent ;
    }

    public int leggiCentesimi ()
    {
        centesimi = Console.readInt("Inserire ora: ");
        return centesimi ;
    }
}
```

- La classe `tempo2` conterrà tutti i metodi e i campi della classe `tempo` ed in più i campi e i metodi definiti in essa.



Proprietà delle classi e degli oggetti Java

Polimorfismo :

Selezione di un metodo tra diversi metodi che hanno lo stesso nome in base al tipo ed al numero dei parametri.

- All'interno di una stessa classe o di una classe da essa derivata si possono avere più metodi con lo stesso nome ma con parametri diversi per tipo e/o per numero.
- Ad esempio, possiamo metodi diversi che effettuano operazioni matematiche su dati di tipo differente. Oppure due funzioni `assegnatempo` in cui la seconda usa la prima per assegnare ore, minuti e secondi e quindi assegna anche i centesimi di secondi.
- L'insieme del nome e dei parametri di un metodo sono detti ***firma del metodo***. **La firma è unica.**
- Non si possono avere metodi che differiscono per il tipo del valore restituito.

Proprietà delle classi e degli oggetti Java

Overriding :

La possibilità di ri-definire (sovrascrivere) in una classe derivata i metodi della classe genitrice.

- In questo caso, quando viene invocato un metodo si usa quello che è definito all'interno della sottoclasse.
- Ad esempio, nella classe `tempo2` si può definire un metodo `assegnaTempo` per assegnare anche i centesimi che ridefinisce il metodo con lo stesso nome della classe `tempo`.

```
miotempo2.assegnatempo(o, m, s, c);
```

- Se invece si vuole usare in una classe derivata un metodo della classe genitrice che ha lo stesso nome del metodo della classe derivata si può fare tramite la parola chiave `super` :

```
super.assegnatempo(o, m, s);
```

- L'overriding non va confuso con il polimorfismo. **Nel caso dell'overriding i metodi hanno stessa firma.** Ma stanno in due classi diverse!

Proprietà delle classi e degli oggetti Java

- I dati delle superclassi che vengono usati da metodi delle classi derivate devono essere dichiarate di tipo **protected non private**.
- Vedere l'esempio della classe **tempo** nel caso in cui si definisca la classe **tempo2** :

```
class tempo {  
    protected int ore;  
    protected int minuti;  
    protected int secondi;  
    public char separatore;  
    . . . . .  
    . . . . .  
}
```

- Se i dati della superclasse vengono modificati, il compilatore potrà segnalare quale parti modificare nelle sottoclassi derivate.