Classi astratte

Sono classi per le quali non è possibile creare oggetti

Rappresentano insiemi costituiti esclusivamente da sottoinsiemi

---

Nota su classi/insiemi

Le classi Java servono per realizzare insiemi di oggetti Java

Le classi vengono spesso usate per rappresentare insiemi di "cose"

Per esempio:

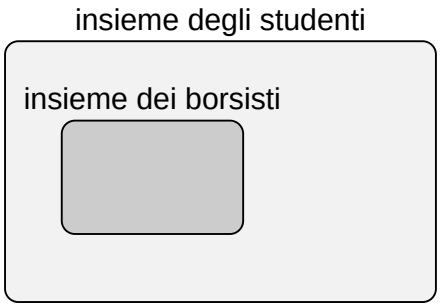
Classe `Studente`:
    insieme di oggetti `Studente`
Classe `Borsista`:
    insieme di oggetti `Borsista`

Dato che ogni oggetto `Studente` rappresenta uno studente, si tende a pensare che la classe `Studente` rappresenta l'insieme degli studenti

Non è esattamente cosí

Insiemi

La rappresentazione grafica di studenti e borsisti è la seguente:



Nota: questi sono gli insiemi di studenti e borsisti, non gli insiemi di oggetti Java

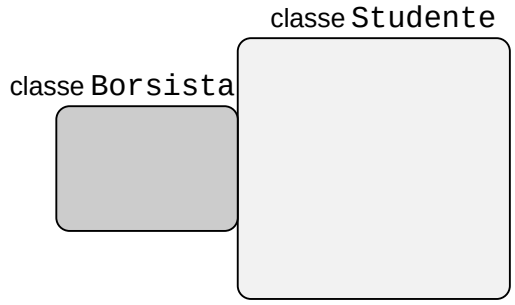
Tutti i borsisti sono anche studenti

Classi Java

In Java, ogni oggetto è istanza di una classe
(si può vedere la classe di un oggetto facendo `oggetto.getClass()`

Un oggetto di una classe non può essere anche oggetto di un'altra classe

Un oggetto `Borsista` è un oggetto della classe `Borsista`
non è un oggetto della classe `Studente`

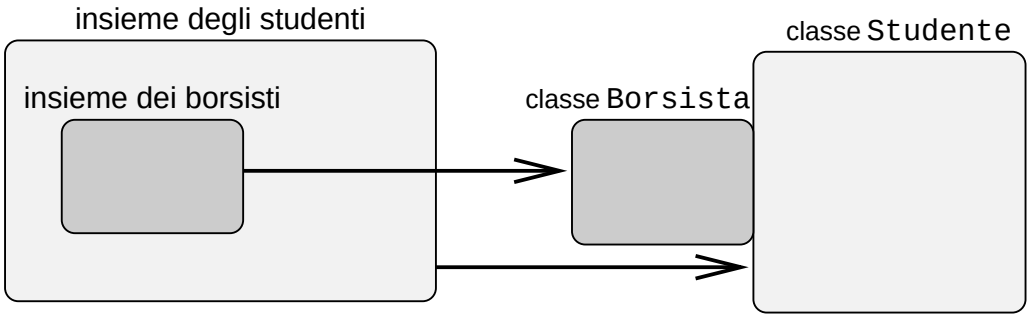


Solo gli oggetti `Studente` sono oggetti della classe `Studente`

Cosa rappresentano gli oggetti

`Borsista`
: un borsista

`Studente`
: uno studente *che non è anche un borsista*



Un borsista è rappresento come un oggetto `Borsista` e non come un oggetto `Studente`

Variabili

In una variabile `Studente` ci si può mettere:

- il riferimento a un oggetto `Studente`
- il riferimento a un oggetto `Borsista`

Quindi, una variabile `Studente` può contenere un oggetto che rappresenta uno studente qualsiasi (sia esso un borsista oppure no)

Classi/Variabili

In generale, se un insieme è rappresentato da una classe, gli oggetti e le variabili della classe rappresentano:

oggetto
: un elemento dell'insieme *che non è anche elemento di uno dei sottoinsiemi*

variabile
: un elemento dell'insieme, inclusi quelli dei sottoinsiemi

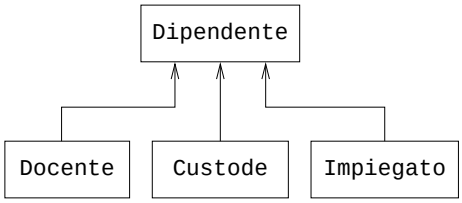
Classi astratte: esempio

Rappresentare le classi relative ai dipendenti dell'università, sapendo che:

1. tutti i dipendenti hanno un numero di matricola
2. i dipendenti possono essere soltanto: docenti, impiegati e custodi;
3. i docenti hanno un dipartimento di afferenza, gli impiegati un ufficio, e i custodi una struttura.

Soluzione

Classe `dipendente` con varie sottoclassi:



Quale specifica può non essere rispettata?

Oggetti della sovraclasse

Quando si estende una classe, questa rimane inalterata.

```
class Dipendente {
  int matricola;
}
```

Quindi, è possibile creare oggetti della classe `Dipendente`.

Questi sarebbero dipendenti che non sono nè docenti, nè impiegati, nè custodi.

Soluzioni errate

Una soluzione errata è quella di realizzare le tre classi `Docente`, `Impiegato` e `Custode` senza che estendano `Dipendente`.

Perchè non va bene:

1. se si vuole aggiungere una componente (es. data di nascita) a tutti questi oggetti, va fatto in tutte le classi;
2. non si possono realizzare metodi che hanno un `Dipendente` generico come argomento.

Classi astratte

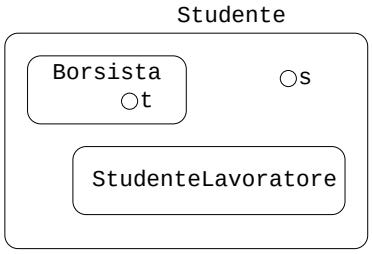
Sono classi dichiarate come `abstract class Nome` invece del solito `class Nome`

tecnicamente
: non si possono creare oggetti della classe

logicamente
: classi i cui oggetti sono solo oggetti delle sottoclassi

Sottoinsiemi

Esistono elementi che non stanno nei sottoinsiemi:



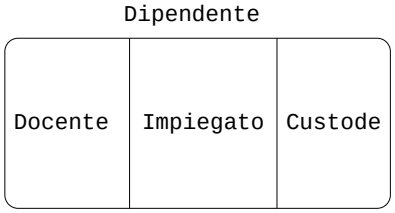
Questo viene modellato usando tutte classi normali.

Gli elementi come `s` sono rappresentati da oggetti `Studente`

Gli elementi come `t` (che sono sia studenti che borsisti) sono rappresentati da oggetti `Borsista`

Partizione

In una partizione, ogni elemento dell'insieme si trova in almeno un sottoinsieme



Si definisce `Dipendente` come classe astratta

Non esistono gli oggetti `Dipendente` ma solo gli oggetti delle sottoclassi

Classi astratte: tecnica

Se una classe è astratta:

1. non si possono creare oggetti della classe
2. per tutto il resto sono classi come le altre:
   - si possono definire variabili della classe
   - in particolare, si possono definire parametri formali del tipo di una classe astratta
   - si possono fare sottoclassi

Classi astratte: definizione

```
abstract class Dipendente {
  int matricola;
}

class Docente extends Dipendente {
  String corso;
}
```

Classi astratte: uso

```
class ProvaDip {
  static void stampaMatricola(Dipendente d) {
    System.out.println(d.matricola);
  }

  public static void main(String args[]) {
    Dipendente d;

    d=new Docente();

    stampaMatricola(d);
  }
}
```

Osservazioni

`Dipendente d;` si può fare

`d=new Docente();` si può fare

`d=new Dipendente();` è l'unica cosa che non si può fare (creare un oggetto di una classe astratta)

```
  static void stampaMatricola(Dipendente d) {
    System.out.println(d.matricola);
  }
```

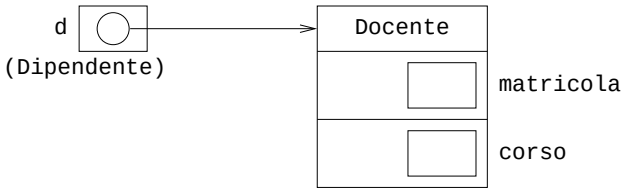
Si può usare `Dipendente` come tipo del parametro formale (è una variabile come tutte le altre)

Si può usare `d.matricola`

A proposito di componenti e metodi

Se `d` è di tipo `Dipendente`
si può usare `d.dipendente` come al solito

Quale componente viene usata?



La variabile è di tipo `Dipendente`, ma l'oggetto non può che essere `Docente` oppure `Impiegato` oppure `Custode`

Tutte queste sono però sottoclassi di `Dipendente`

Quindi la componente `matricola` c'è in tutte.

Precisazione su estensioni e componenti

Quello che permette di fare `Classe c=new Sottoclasse();` è il fatto che la sottoclasse è stata dichiarata con `extends`

Da questo deriva che:

1. componenti e metodi vengono ereditati
2. si può fare l'assegnamento

Il punto 1 garantisce che il punto 2 non crea problemi (`c.componente_classe` funazione)

Però non vale il viceversa

Compatibilità per struttura

Questo si può fare:

```
Dipendente d=new Docente();
```

Questo però no:

```
class Point3D {
  int x;
  int y;
  int z;
}

...
  // errore
Point p=new Point3D();
```

Anche se `Point3D` ha tutte le componenti di `Point`, non è una sottoclasse.

Ereditare le componenti (e i metodi) e la validità dell'assegnemento sono due **conseguenze** dello stesso fatto (usare `extends`)

La seconda non è una conseguenza della prima.

Metodi astratti

Una classe astratta può contenere componenti e metodi

Può anche contere un metodo astratto, che è una definizione di metodo senza implementazione

```
abstract class Astratta {
  int x;

                // metodo non astratto
  int esempio(int x) {
    return x*2;
  }

                // metodo astratto
  abstract int altro(int x);
}
```

A cosa servono i metodi astratti

Tutte le sottoclassi (non astratte) devono implementare i metodi astratti

Quindi, per ogni metodo della classe astratta:

1. la firma è definita nella classe astratta;
2. il corpo del metodo è definito nella sottoclasse

Quindi, per ogni variabile `x` della classe astratta, `x.metodo_astratto(...)` si può fare, e si sa esattamente cosa succede

---

A cosa servono i metodi astratti

Esempio: stampare i dati di un dipendente

```
abstract class Dipendente {
  int matricola;

  abstract void stampa();
}


class Docente extends Dipendente {
  String corso;
```

```
  void stampa() {
    System.out.println(matricola+" "+corso);
  }
}
```

In questo modo, si definiscono metodi che possono essere invocati su una variabile `Dipendente`, ma che sono definiti effettivamente solo nelle sottoclassi

---

Sottoclassi astratte

Si possono realizzare sottoclassi astratte di una classe astratta

Si possono anche realizzare sottoclassi astratte di una classe non astratta