

Appunti di Progettazione del Software

Canale 1 · Prof. Paolo Liberatore

Capitolo 5: Classi astratte

Autore: Tommaso C.

Appunti tratti dalle lezioni in aula e dal materiale del corso

Argomenti: Classi astratte · Variabili e oggetti come insiemi
Metodi astratti · Costruttori di una classe astratta

Indice

1	La classe astratta	1
2	Metodi astratti	2
3	Il costruttore di una classe astratta	3
4	Esercizi	4
5	Soluzioni	5

1 La classe astratta

Definizione – Classe astratta

Una classe dichiarata **abstract class C** non ha oggetti propri: `new C(...)` è un errore di compilazione. I suoi oggetti sono solo quelli delle sottoclassi, e una variabile di tipo `C` può contenere un riferimento a ciascuno di essi.

La classe **Figura** del Capitolo 3 serve a raccogliere rettangoli, triangoli e quadrati in un `HashSet<Figura>` e a scorrerli con un solo ciclo. Una figura generica, però, non ha senso: nell'Esercizio 1 del Capitolo 3 il suo perimetro è uno stub che restituisce un valore fittizio. Dichiarandola astratta, figure generiche non se ne creano più:

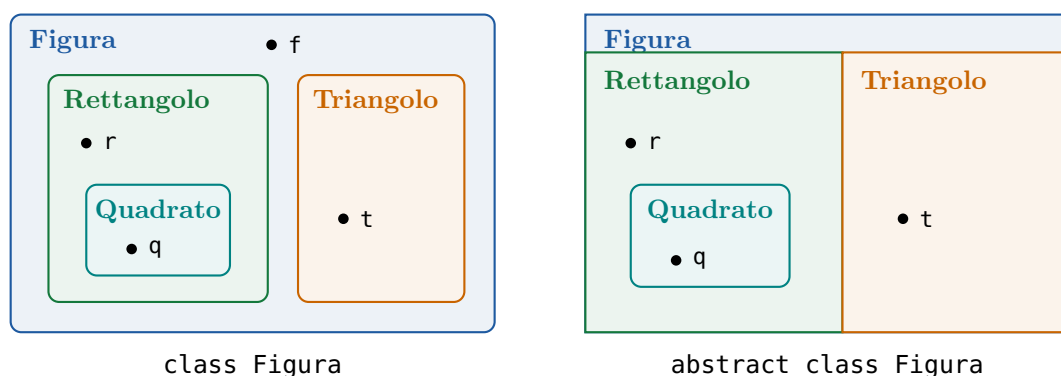
```
public abstract class Figura {
    // campi e costruttore come prima
}
```

Ora `new Figura("generica")` non compila: *Figura is abstract; cannot be instantiated*. Resta valida `Figura f = new Quadrato("piastrella", 2);`.

Variabili e oggetti come insiemi. Una classe si può vedere come un insieme che contiene quelli delle sue sottoclassi. Variabili e oggetti ne occupano parti diverse:

- una **variabile** di tipo **Rettangolo** può riferire qualunque oggetto dell'insieme dei rettangoli, quadrati compresi;
- un **oggetto** creato con `new Rettangolo(...)` sta nei rettangoli ma fuori dai quadrati, nella “cornice” fra i due insiemi.

Dichiarare una classe astratta svuota la sua cornice: gli insiemi delle sottoclassi la **ripartiscono** completamente.



*I punti sono oggetti: f creato con `new Figura`, r con `new Rettangolo`, q con `new Quadrato`, t con `new Triangolo`. A destra la cornice di **Figura** è vuota: ogni figura è un rettangolo o un triangolo.*

Osservazione – Una partizione delle classi definite

La partizione riguarda le sottoclassi scritte nel programma. Con **Rettangolo** e **Triangolo**, le figure rappresentabili sono rettangoli e triangoli; aggiungendo una classe **Ellisse** `extends Figura`, la partizione passa da due a tre parti.

Osservazione – Astratta o no: lo dicono i requisiti

Se una classe sia astratta dipende dall'applicazione. In una banca che gestisce clienti e consulenti, una persona che non sia né l'uno né l'altro probabilmente non interessa, e **Persona** sarà astratta. Ma se interessano anche impiegati o direttori, la domanda va fatta agli esperti del dominio (Capitolo 1): danno spesso per scontato proprio questo genere di informazioni. Esercizio 1.

2 Metodi astratti

Definizione – Metodo astratto

Un **metodo astratto** ha solo l'intestazione, preceduta da **abstract** e chiusa da un punto e virgola, senza corpo:

- si può dichiarare solo in una classe astratta;
- ogni sottoclasse non astratta deve averne una definizione, scritta in essa o ereditata.

Il metodo astratto dice: *tutte le figure hanno un perimetro, ma non c'è modo di calcolarlo senza sapere che figura sia*. Ogni sottoclasse lo calcola a modo suo.

```
public abstract class Figura {
    String descrizione;

    Figura(String descrizione) {
        this.descrizione = descrizione;
    }

    abstract double perimetro();

    abstract double area();
}
```

Le sottoclassi rispettano già l'obbligo: **Rettangolo** e **Triangolo** definiscono `perimetro()` e `area()`, e **Quadrato** eredita quelli di **Rettangolo**, che per un quadrato sono giusti. Se

Triangolo non avesse `area()`, il compilatore darebbe *Triangolo is not abstract and does not override abstract method area() in Figura*.

Il perimetro totale ora si scrive senza stub:

```
static double perimetroTotale(HashSet<Figura> figure) {
    double totale = 0;
    for (Figura f : figure) {
        totale = totale + f.perimetro();
    }
    return totale;
}
```

`f.perimetro()` compila perché **Figura** dichiara il metodo; si esegue quello della classe dell'oggetto. Con il rettangolo 4×5 , il triangolo di lati 3, 2, 4 e il quadrato di lato 2 il totale è $18 + 9 + 8 = 35$.

Nota – Stub o metodo astratto

	Stub in Figura	Metodo astratto
corpo in Figura	valore fittizio	nessuno
<code>new Figura(...)</code>	ammesso	errore di compilazione
sottoclasse senza il metodo	eredita il valore fittizio	errore di compilazione

3 Il costruttore di una classe astratta

Proposizione – Costruttore di una classe astratta

Una classe astratta può avere costruttori. Non li chiama `new`, ma i costruttori delle sottoclassi, con `super(...)`: inizializzano la parte comune a tutti gli oggetti.

Nel costruttore di **Figura** si può quindi mettere un controllo che vale per tutte le figure:

```
Figura(String descrizione) {
    if (descrizione == null || descrizione.isEmpty()) {
        throw new RuntimeException("Descrizione vuota");
    }
    this.descrizione = descrizione;
}
```

`new Rettangolo("", 1, 2)` chiama `super("")` e lancia l'eccezione: nessuna figura nasce con la descrizione vuota. Perché non la riceva dopo, `descrizione` va anche dichiarato `private` (Capitolo 4).

Riepilogo

- **Classe astratta:** `abstract class C; new C(...)` non compila, gli oggetti sono quelli delle sottoclassi.
- **Variabili:** una variabile di tipo `C` vale anche per una classe astratta e riferisce oggetti delle sottoclassi.
- **Come insiemi:** un oggetto di `C` sta nella cornice di `C`; se `C` è astratta la cornice è vuota e le sottoclassi la ripartiscono.
- **Requisiti:** se una classe sia astratta lo decide l'applicazione; nel dubbio si chiede agli esperti del dominio.
- **Metodo astratto:** intestazione senza corpo, solo in una classe astratta; ogni sottoclasse non astratta lo definisce o lo eredita.
- **Senza stub:** il metodo astratto si chiama su una variabile della superclasse e si esegue quello della classe dell'oggetto.
- **Costruttore:** una classe astratta può averne uno, che i costruttori delle sottoclassi chiamano con `super(...)`.

4 Esercizi

Esercizio 1 – La banca

Una banca vuole gestire clienti, consulenti, conti e appuntamenti.

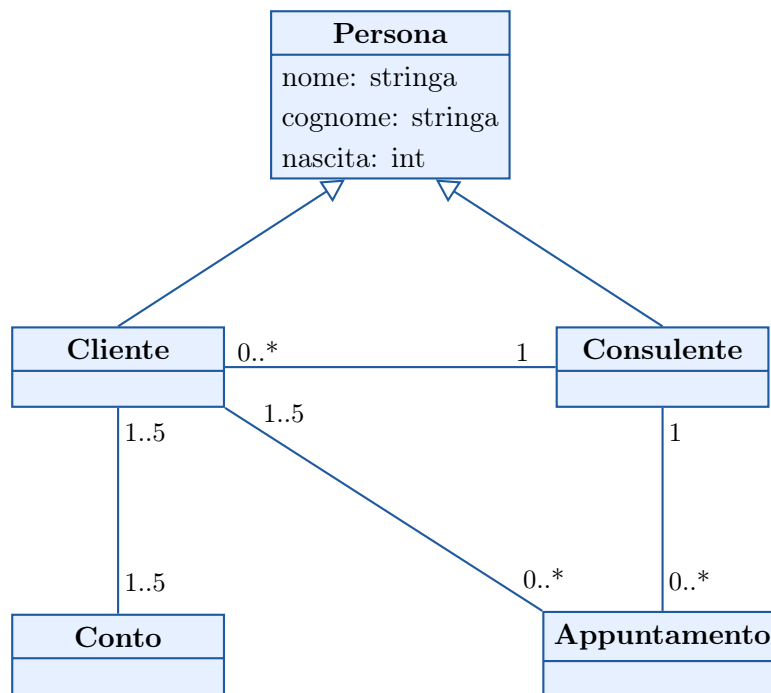
- Di ogni persona interessano nome, cognome e anno di nascita; le uniche persone di interesse sono clienti e consulenti.
 - Ogni cliente ha esattamente un consulente; un consulente può avere un numero qualsiasi di clienti.
 - Ogni cliente è titolare di un numero di conti da uno a cinque, e ogni conto ha da uno a cinque titolari.
 - Ogni appuntamento è con un solo consulente e riguarda da uno a cinque clienti; clienti e consulenti possono avere un numero qualsiasi di appuntamenti.
- (a) Disegnare il diagramma delle classi.
- (b) Scrivere le classi Java, con le associazioni navigabili nei due versi, senza imporre i vincoli di molteplicità.

Soluzione a pagina 5 →

5 Soluzioni

Soluzione dell'Esercizio 1 – La banca

(a) Clienti e consulenti sono persone: **Cliente** e **Consulente** sono sottoclassi di **Persona**, con una freccia ciascuna. Conti e appuntamenti non sono persone, né l'una né l'altra cosa: sono classi a sé, collegate con linee.



La molteplicità si legge all'estremo opposto: l'1 vicino a **Consulente** dice che un cliente ha un consulente, lo 0..* vicino a **Cliente** che un consulente ha un numero qualsiasi di clienti. Una classe **Banca** non serve: la banca è una sola, e una classe modella un insieme di oggetti.

(b) **Persona** è astratta: le sue istanze sono clienti o consulenti.

```

public abstract class Persona {
    String nome;
    String cognome;
    int nascita;

    Persona(String nome, String cognome, int nascita) {
        this.nome = nome;
        this.cognome = cognome;
        this.nascita = nascita;
    }
}

public class Consulente extends Persona {

```

```

HashSet<Cliente> clienti;
HashSet<Appuntamento> appuntamenti;

Consulente(String nome, String cognome, int nascita) {
    super(nome, cognome, nascita);
    this.clienti = new HashSet<Cliente>();
    this.appuntamenti = new HashSet<Appuntamento>();
}
}

```

Consulente non ha dati propri, ma ha comunque un costruttore: **Persona** ha solo un costruttore con argomenti (Capitolo 3). Cliente e consulente differiscono per le associazioni, non per i dati.

```

public class Cliente extends Persona {
    Consulente consulente;
    HashSet<Conto> conti;
    HashSet<Appuntamento> appuntamenti;

    Cliente(String nome, String cognome, int nascita,
            Consulente consulente) {
        super(nome, cognome, nascita);
        this.consulente = consulente;
        consulente.clienti.add(this); // l'altro verso
        this.conti = new HashSet<Conto>();
        this.appuntamenti = new HashSet<Appuntamento>();
    }
}

public class Conto {
    HashSet<Cliente> titolari;

    Conto() {
        this.titolari = new HashSet<Cliente>();
    }
}

public class Appuntamento {
    Consulente consulente;
    HashSet<Cliente> clienti;

    Appuntamento(Consulente consulente) {
        this.consulente = consulente;
        consulente.appuntamenti.add(this);
        this.clienti = new HashSet<Cliente>();
    }
}

```

```
}  
}
```

Le classi con `HashSet` importano `java.util.HashSet`. Il consulente di un cliente e quello di un appuntamento sono campi singoli, perché il massimo è 1; tutti gli altri estremi sono insiemi. Il costruttore che riceve il consulente aggiorna anche l'altro verso, così i due collegamenti restano coerenti (Capitolo 2).

```
Consulente c = new Consulente("Laura", "Verdi", 1975);  
Cliente x = new Cliente("Mario", "Rossi", 1980, c);  
Cliente y = new Cliente("Anna", "Bianchi", 1992, c);  
Conto k = new Conto();  
k.titolari.add(x);  
x.conti.add(k);  
k.titolari.add(y);  
y.conti.add(k);  
Appuntamento a = new Appuntamento(c);  
a.clienti.add(x);  
x.appuntamenti.add(a);  
// c.clienti.size() = 2, k.titolari.size() = 2  
// x.consulente.cognome = "Verdi"
```

Un conto e un cliente nascono senza collegamenti, e i vincoli 1..5 non sono garantiti: un conto senza titolari o con sei titolari si costruisce senza errori. Per imporli servono campi `private` e metodi che controllano; per gli insiemi un getter non basta (Capitolo 4).

[← Torna all'Esercizio 1, pagina 4](#)