DIIES Dipartimento di
INGEGNERIA
dell'INFORMAZIONE, delle INFRASTRUTTURE e dell'ENERGIA SOSTENIBILE

Corso di Fondamenti di Informatica

Dispensa 9: Composizione di Classi

Prof. Domenico Rosaci

2015-16

Composizione di Classi

In Java, la dichiarazione di un oggetto appartenente ad una determinata classe, comporta semplicemente la creazione di un riferimento all'oggetto e non la costruzione in memoria dell'oggetto stesso.

Esempio:

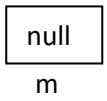
Sia Materia una classe che rappresenti una materia universitaria, e che contenga due campi, il primo rappresentante il nome della materia e il secondo rappresentante il settore scientifico disciplinare a cui la materia afferisce.

```
class Materia {
        private String denominazione;
        private String ssd;
        public Materia(String d, String s){
                denominazione=d;
                ssd=s;
        }

public Materia(Materia m){
                denominazione=m.getDenominazione();
ssd=m.getSsd());
        }

public String getDenominazione(){
                return denominazione;
        }
public void setDenominazione(String s){
                denominazione=s;
        }

        public String getSsd(){
                return ssd;
        }
        public Materia getStatus(){
                Materia m=new Materia(denominazione,ssd);
                return m;
        }
        public void stampa(){
                System.out.println(denominazione+"("+ssd+")");
        }
}
```

Si consideri la seguente istruzione, che dichiara una variabile **m** della classe *Materia*.

Materia m;

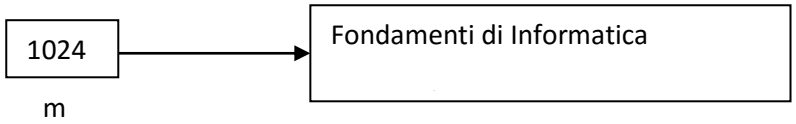
la situazione che si crea con questa istruzione corrisponde alla creazione di una cella che contiene un indirizzo di memoria (un riferimento) ad un oggetto di classe *Materia*. Tuttavia, poiché l'oggetto non è stato ancora creato, il riferimento contenuto nella variabile non corrisponde a nessuna cella di memoria, e viene indicato con l'identificatore *null.*



Se si vuole far “puntare” **m** ad un oggetto della classe *Materia*, bisogna creare l’oggetto usando l’operatore new abbinato ad un costruttore della classe.

m=new Materia(“Fondamenti di Informatica”, “ING-INF/05”);

In seguito a questa istruzione, la variabile **m** “punta” all’oggetto appena creato.



Nell’esempio di cui sopra, abbiamo supposto che l’oggetto a cui **m** punta sia stato creato all’indirizzo di memoria 1024. L’insieme dei due valori “Fondamenti di Informatica” e “ING-INF/05” rappresenta lo *stato* dell’oggetto **m**.

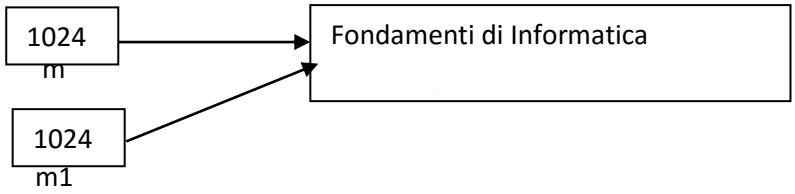
Adesso immaginiamo di voler creare una copia di **m**, ed a tale scopo dichiariamo una seconda variabile di tipo Materia, ad esempio:

Materia m1;

Per ottenere che **m**1 abbia lo stesse stato di **m**, potremmo scrivere:

m1=m;

Tuttavia, anche se così **m1** assume lo stesso stato di **m**, esso non rappresenta una copia dell’oggetto a cui **m** fa riferimento. Semplicemente, **m1** è un secondo riferimento all’oggetto puntato da **m**.



L’operazione che abbiamo effettuato è potenzialmente “pericolosa”. Infatti, se operiamo modifiche sull’oggetto attraverso **m1**, dobbiamo essere coscienti che l’oggetto modificato è lo stesso a cui anche **m** fa riferimento. Ad esempio, se modifichiamo la denominazione dell’oggetto puntato da **m1** scrivendo:

m1.setDenominazione(“Chimica”);

possiamo riscontrare il cambiamento di denominazione anche attraverso **m**, scrivendo:

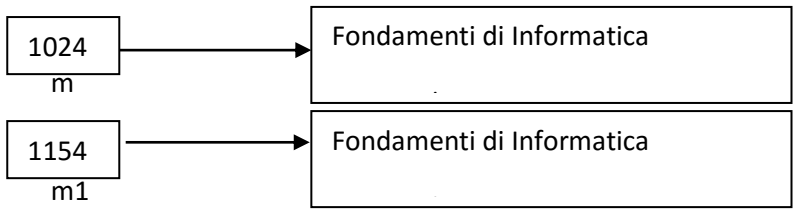
System.out.println(m.getDenominazione());

avendo come effetto la visualizzazione sullo schermo della stringa “Chimica”.

Per questo motivo, al fine di evitare tale effetto collaterale degli assegnamenti di riferimenti, se vogliamo creare una copia di un oggetto dobbiamo usare new ed il costruttore:

m1=new Materia(m.getDenominazione(),m.getSsd());

Ottenendo la seguente situazione:



che produce un'effettiva copia dell'oggetto puntato da **m**, assegnando tale copia al riferimento **m1**.

Possiamo ottenere una copia dell'oggetto puntato da **m** anche attraverso il *costruttore di copia* della classe *Materia*. Questo costruttore accetta in ingresso un oggetto della classe, e crea un nuovo oggetto avente lo stesso stato dell'oggetto passato. Quindi l'istruzione:
m1=new Materia(m);

produce lo stesso effetto dell'istruzione

m1=newMateria(m.getDenominazione(),m.getSsd());

Infine, un terzo modo di creare una copia dell'oggetto puntato da **m** è quello di usare un metodo getStatus() della classe *Materia*, che appunto crea un oggetto di classe *Materia* avente lo stesso stato dell'oggetto che chiama il metodo getStatus(), e restituisce un riferimento a questo nuovo oggetto.

m1=m.getStatus();

Ovviamente questo terzo metodo di creare una copia dell'oggetto, rispetto all'uso del costruttore di copia, consente di assegnare la copia ad un nuovo riferimento in qualunque momento, e non solo in fase di creazione del riferimento.

Le considerazioni di cui sopra sono particolarmente importanti quando si introduce la "Composizione di classi", ovvero quando si dichiara una classe in cui alcuni campi sono oggetti di altre classi. E' questo il caso, ad esempio, della classe *Corso* illustrata di seguito, composta da un campo **materia** di classe *Materia*, da un campo intero **anno**, e da un campo **doc** di classe *Docente*.

```
class Docente {
        private String nome;
        private String cognome;
        public Docente(String n, String c){
                nome=n;
                cognome=c;
```

```
	}
	public String getNome(){
		return nome;
	}
	public String getCognome(){
		return cognome;
	}
	public Docente getStatus(){
		Docente d=new Docente(nome,cognome);
		return d;
	}
	public void stampa(){
		System.out.println(nome+" "+cognome);
	}
}


class Corso {
	private Materia materia;
	private int anno;
private Docente doc;

	public Corso(Materia m, int a, Docente d){
		materia=m.getStatus();
		anno=a;
		doc =d.getStatus();
	}
	public Materia getMateria(){
		return materia.getStatus();
	}
	public int getAnno(){
		return anno;
	}
public Docente getDocente(){
		return doc.getStatus();
	}

	public Corso getStatus(){
		Corso e=new Corso(materia,anno,doc);
		return e;
	}
	public void stampa(){
		materia.stampa();
		System.out.println("anno:"+anno);
doc.stampa();

	}
}
```

Come si vede, il costruttore parametrizzato della classe *Corso*, per assegnare al campo **materia** lo stato dell'oggetto a cui fa riferimento il parametro **m**, non effettua una semplice assegnazione di riferimento, del

tipo **materia=m**, ma invece assegna a **materia** una copia dell'oggetto **m**, ottenuta invocando **m**.getStatus. Ciò consente di creare un oggetto di classe *Corso* il cui campo **materia** è reso indipendente dall'oggetto **m** di classe *Materia* usato per crearlo. Per presentare un esempio concreto in questo contesto, immaginiamo di creare un oggetto di classe *Materia* e un oggetto di classe *Docente* nel modo seguente:

Materia m=new Materia("Fondamenti di Informatica", "ING-INF/05");
Docente d=new Docente("Mario","Rossi");

adesso serviamoci di **m** e di **d** per creare un oggetto della classe *Corso*:

Corso c=new Corso(m,2008,d);

se successivamente noi volessimo cambiare lo stato dell'oggetto **m**, ad esempio cambiando la denominazione della materia in "Reti di Calcolatori", tale modifica non si rifletterà sullo stato dell'oggetto **c**, che continuerà a contenere un riferimento alla materia "Fondamenti di Informatica". Questo comportamento "corretto" è stato ottenuto procedendo come descritto sopra nel costruttore parametrizzato della classe *Corso*. Se in questo costruttore avessimo scritto, invece di **materia**=**m**.getStatus, semplicemente **materia**=**m**, la modifica sull'oggetto **m** avrebbe comportato l'effetto collaterale di una modifica sullo stato dell'oggetto **c**.

Per identico motivo, anche i metodi getMateria() e getDocente() della classe *Corso*, non restituiscono dei riferimenti ai corrispondenti campi dell'oggetto, ma piuttosto riferimenti a *copie* di questi campi. Infatti, supponiamo come esempio che il metodo getMateria() di *Corso* restituisse il campo **materia** dell'oggetto (usando l'istruzione return **materia** invece di return **materia**.getStatus()). Allora l'istruzione:

Materia v=c.getMateria();

permetterebbe di creare un riferimento **v** all'oggetto che rappresenta la materia del corso **c**. Attraverso questo riferimento si potrebbe modificare lo stato del corso **c**, ad esempio scrivendo:

v.setDenominazione("Reti di Calcolatori");

Fortunatamente, getMateria() restituisce solo una copia dell'oggetto che rappresenta la materia del corso **c**, impedendo tale inconveniente.

Di seguito sono riportate le descrizioni di altre due classi, *Esame* e *Studente*, che rappresentano due ulteriori esempi di composizione tra classi.

```
class Esame {
        private Corso corso;
        private int voto;
        public Esame(Corso c, int v){
                corso=c.getStatus();
                voto=v;
        }
        public Corso getCorso(){
                return corso.getStatus();
        }
        public int getVoto(){
                return voto;
        }
        public Esame getStatus(){
                Esame e=new Esame(corso,voto);
```

```
		return e;
	}
	public void stampa(){
		corso.stampa();
		System.out.println("voto:"+voto);
	}
}


public class Studente {
	private String nome;
	private String cognome;
	private int matricola;
	private Esame[] listaEsami;
	public Studente(String n, String c, int m, Esame e[]){
		nome=n;
		cognome=c;
		matricola=m;
		listaEsami=new Esame[e.length];
		for(int i=0;i<e.length;i++)
			listaEsami[i]=e[i].getStatus();
	}
	public void stampa(){
		System.out.println(nome+" "+cognome+", matricola:"+matricola);
		for(int i=0;i<listaEsami.length;i++)
			listaEsami[i].stampa();
	}
	public static void main(String[] args){
		Docente profRossi=new Docente("Mario","Rossi");
		Docente profBianchi=new Docente("Gianni","Bianchi");
		Materia Informatica=new Materia("Informatica","ING-INF/05");
		Materia Analisi=new Materia("Analisi","MAT/05");
		Corso Informatica08=new Corso(Informatica,2008,profRossi);
		Corso Analisi08=new Corso(Analisi,2008,profBianchi);
		Esame[] esamiPasquale=new Esame[2];
		esamiPasquale[0]=new Esame(Informatica08,28);
		esamiPasquale[1]=new Esame(Analisi08,25);
		Studente pasquale=new Studente("Pasquale","Verdi",126,esamiPasquale);
		pasquale.stampa();

	}
}
```