Metodologie di Programmazione

Lezione 15: Polimorfismo

Lezione 15: Sommario

UNITELMA SAPIENZA
SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Polimorfismo
- Esempi e casistiche
- La parola chiave super per metodi e campi
- Operatore instanceof
- Downcasting e upcasting

Polimorfismo

- Insieme all'ereditarietà, un altro concetto cardine della programmazione orientata agli oggetti
- Polimorfismo = **molte** + **forme**

1. Una variabile di un certo tipo A può contenere un riferimento a un oggetto del tipo A o di qualsiasi sua sottoclasse

```
Animale a = new Gatto();
a = new Chihuahua();
```

2. La selezione del metodo da chiamare avviene in base all'effettivo tipo dell'oggetto riferito dalla variabile

```
Animale a = new Gatto();
a.emettiVerso();
a = new Chihuahua();
a.emettiVerso();
```

Miaooo!

Bau bau!

Binding statico vs. dinamico

- Il binding statico consiste nell'associare un metodo al tipo della classe di una variabile riferimento
  - Questo non permetterebbe di chiamare il metodo appropriato
- Il polimorfismo implementa invece il binding dinamico, poiché l'associazione tra una variabile riferimento e il metodo viene stabilita a tempo di esecuzione

```
Animale a = new Gatto();
a = new Chihuahua();
```

Chiamare metodi della superclasse

- E' sufficiente utilizzare la parola chiave super:

```java
import java.util.Random;

public class StringaHackerata
{
    private String s;

    public StringaHackerata(String s)
    {
        this.s = s;
    }

    @Override
    public String toString()
    {
        StringBuffer sb = new StringBuffer();
        Random random = new Random();

        for (int k = 0; k < s.length(); k++)
        {
            char c = s.charAt(k);
            if (random.nextBoolean()) c = Character.toUpperCase(c);
            else c = Character.toLowerCase(c);

            sb.append(c);
        }

        return sb.toString();
    }
}
```

```java
import java.util.Random;


public class StringaHackerataConStriscia extends StringaHackerata
{
    final public static int MAX_LUNGHEZZA = 10;

    public StringaHackerataConStriscia(String s)
    {
        super(s);
    }

    public String getStriscia()
    {
        Random random = new Random();
        int len = random.nextInt(MAX_LUNGHEZZA);
        StringBuffer sb = new StringBuffer();

        // -=-=-=-
        for (int k = 0; k < len; k++) sb.append(k % 2 == 0 ? '-' : '=');

        return sb.toString();
    }

    @Override
    public String toString()
    {
        String striscia = getStriscia();
        return striscia+" "+super.toString()+" "+striscia;
    }
}
```

Chiamiamo il metodo della superclasse

StringaHackerata e StringaHackerataConStriscia

- Eseguendo il metodo main:

```
public static void main(String[] args)
{
    StringaHackerata s1 = new StringaHackerata("drago della programmazione");
    StringaHackerataConStriscia s2 = new StringaHackerataConStriscia("drago della programmazione");

    System.out.println(s1);
    System.out.println(s2);
}
```

- Si ottiene:

```
DraGo DElla PROgraMMaZioNE
-=-=-=-=- dRAgO DelLA PROGRAMmaZiONe -=-=-=-=-
```

- E se scrivo...?

```
public static void main(String[] args)
{
    StringaHackerata s1 = new StringaHackerata("drago della programmazione");
    StringaHackerata s2 = new StringaHackerataConStriscia("drago della programmazione");

    System.out.println(s1);
    System.out.println(s2);
}
```

Stesso risultato grazie al **polimorfismo**!

Esempio: Modellare gli impiegati (1)

- La classe Impiegato modella il nome e il codice dell'impiegato
- Permette di aggiornare il nome dell'impiegato
- Restituisce su richiesta il codice (id) e il nome
- Definisce il metodo toString() restituendo la stringa "nome (codice)"

```
public class Impiegato
{
    /**
     * Nome dell'impiegato
     */
    private String nome;

    /**
     * Identificativo dell'impiegato
     */
    private String id;

    public Impiegato(String nome, String id)
    {
        this.nome = nome;
        this.id = id;
    }

    /**
     * Aggiorna il nome dell'impiegato
     */
    public void setNome(String nome)
    {
        this.nome = nome;
    }

    public String getId() { return id; }
    public String getNome() { return nome; }

    @Override
    public String toString()
    {
        return nome+" ("+id+")";
    }
}
```

Esempio: Modellare gli impiegati (2)

- Modelliamo ora la classe ImpiegatoStipendiato
- Ha un suo stipendio mensile
- Sovrascrive toString() riutilizzando il toString della superclasse e aggiungendo lo stipendio

```
public class ImpiegatoStipendiato extends Impiegato
{
    private double stipendio;

    public ImpiegatoStipendiato(String nome, String id, double stipendio)
    {
        super(nome, id);
        this.stipendio = stipendio;
    }

    public double getStipendio() { return stipendio; }

    public void setStipendio(double nuovoStipendio) { stipendio = nuovoStipendio; }

    @Override
    public String toString()
    {
        return super.toString()+": "+stipendio+" euro";
    }
}
```

Esempio: Modellare gli impiegati (3)

- Modelliamo infine la classe ImpiegatoACottimo
- E' connotato dalla paga per prodotto e dal numero di prodotti lavorati

```
public class ImpiegatoACottimo extends Impiegato
{
    private double pagaPerProdotto;
    private int prodottiLavorati;

    public ImpiegatoACottimo(String nome, String id, double pagaPerProdotto, int prodottiLavorati)
    {
        super(nome, id);
        this.pagaPerProdotto = pagaPerProdotto;
        this.prodottiLavorati = prodottiLavorati;
    }

    public double getPagaPerProdotto() { return pagaPerProdotto; }
    public double getProdottiLavorati() { return prodottiLavorati; }

    public void setPagaPerProdotto(double pagaPerProdotto) { this.pagaPerProdotto = pagaPerProdotto; }
    public void setProdottiLavorati(int prodottiLavorati) { this.prodottiLavorati = prodottiLavorati; }

    @Override
    public String toString()
    {
        return super.toString()+": "+prodottiLavorati+" prodotti * "+pagaPerProdotto+" euro a prodotto";
    }
}
```

Esempio: Modellare gli impiegati (4)

- Testiamo le classi:

```
public class TestaImpiegati
{
    public static void main(String[] args)
    {
        Impiegato i1 = new ImpiegatoStipendiato("Mario", "imp1", 1500);
        Impiegato i2 = new ImpiegatoACottimo("Luigi", "imp2", 10, 5);

        System.out.println(i1);
        System.out.println(i2);
    }

}
```

- Ottenendo questo output:

```
Mario (imp1): 1500.0 euro
Luigi (imp2): 50 prodotti * 10.0 euro a prodotto
```

L'operatore instanceof

- L'operatore, applicato a un oggetto e a un nome di classe, restituisce true se l'oggetto è un tipo o un sottotipo di quella classe
- Ad esempio:

```
public class TestaImpiegati
{
    public static void main(String[] args)
    {
        Impiegato i1 = new ImpiegatoStipendiato("Mario", "imp1", 1500);
        Impiegato i2 = new ImpiegatoACottimo("Luigi", "imp2", 10, 50);

        System.out.println(i1);
        System.out.println(i2);

        System.out.println(i1 instanceof Impiegato);
        System.out.println(i1 instanceof ImpiegatoStipendiato);
        System.out.println(i1 instanceof ImpiegatoACottimo);
    }

}
```

- Stampa:

```
true

true

false
```

Conversione di tipo fra sottoclasse e superclasse

- Posso sempre convertire senza cast esplicito un sottotipo a un supertipo (upcasting)

```
ImpiegatoStipendiato is1 = new ImpiegatoStipendiato("Mario", "imp1", 1500);
Impiegato i = is1;
```

- A volte può essere necessario convertire un supertipo a un sottotipo (downcasting)
- Richiede un cast esplicito

```
ImpiegatoStipendiato is2 = (ImpiegatoStipendiato)i;
```

Un esempio completo con instanceof e downcasting

- Implementiamo un metodo equals di confronto tra un oggetto di tipo SitoWeb e un altro oggetto qualsiasi:

```
public class SitoWeb
{
    private String URL;

    public SitoWeb(String URL)
    {
        this.URL = URL;
    }

    public String getURL() { return URL; }

    @Override
    public boolean equals(Object o)
    {
        if (o == null) return false;

        // uso di instanceof
        if (!(o instanceof SitoWeb)) return false;

        // downcasting
        SitoWeb s = (SitoWeb)o;
        return URL.equals(s.URL);
    }
}
```

Che succede all'interfaccia con la conversione di tipo?

- Con l'upcasting, si "restringe" **temporaneamente** l'interfaccia dell'oggetto alla superclasse:

```
public class ImpiegatoStipendiato extends Impiegato
{
      // ...

      public void setNome(String nome);
      public String getId();
      public String getNome();

      public String toString();

      public double getStipendio();
      public void setStipendio(double nuovoStipendio);
}
```

Interfaccia pubblica di Impiegato

Ulteriori specificazioni di ImpiegatoStipendiato

- ImpiegatoStipendiato is = new ImpiegatoStipendiato(…);

Che succede all'interfaccia con la conversione di tipo?

- Con l'upcasting, si "restringe" **temporaneamente** l'interfaccia dell'oggetto alla superclasse:

```
public class ImpiegatoStipendiato extends Impiegato
{
    // ...

    public void setNome(String nome);
    public String getId();
    public String getNome();

    public String toString();

    public double getStipendio();
    public void setStipendio(double nuovoStipendio);
}
```

Vedo solo i membri di Impiegato

Interfaccia pubblica di Impiegato

Ulteriori specificazioni di ImpiegatoStipendiato

- ImpiegatoStipendiato is = new ImpiegatoStipendiato(…);
- Impiegato i = is;
- Grazie al polimorfismo, chiamando toString() viene comunque chiamata l'implementazione più specifica (quella di ImpiegatoStipendiato)

Che succede all'interfaccia con la conversione di tipo?

- Con l'upcasting, si "restringe" **temporaneamente** l'interfaccia dell'oggetto alla superclasse:

```
public class ImpiegatoStipendiato extends Impiegato
{
    // ...

    public void setNome(String nome);
    public String getId();
    public String getNome();

    public String toString();

    public double getStipendio();
    public void setStipendio(double nuovoStipendio);
}
```

Interfaccia pubblica di Impiegato

Ulteriori specificazioni di ImpiegatoStipendiato

Torno a vedere Tutti i membri di ImpiegatoStipendiato

- ImpiegatoStipendiato is = new ImpiegatoStipendiato(…);
- Impiegato i = is;
- is = (ImpiegatoStipendiato)i;