

Gerarchia di classi Java

Fondamenti di architettura e programmazione
Corso di laurea in Comunicazione digitale

Specializzare un modello

- Una classe modella un pezzo di realtà
 - Una macchina, una moneta, un impiegato
- Spesso è necessario specializzare la versione originale del modello
 - Es: un dirigente è un particolare impiegato
- Occorre evitare la duplicazione dei dati
- Alcuni metodi sono utili per entrambe le classi
- **Specializzare le classi e l'ereditarietà**

AA 2008/09
© M.A. Alberti

2

Fondamenti di architettura e programmazione
Gerarchie di classi

```
public class Impiegato {
    public String nome = "";
    public double stipendio;
    public Date dataNascita;

    public String toString() {...}
}

public class Dirigente {
    public String nome = "";
    public double stipendio;
    public Date dataNascita;
    public String divisione;

    public String toString() {...}
}
```

AA 2008/09
© M.A. Alberti

3

Fondamenti di architettura e programmazione
Gerarchie di classi

Ereditarietà

- Consiste nel definire una classe tramite un'altra già definita
- In Java, per stabilire una relazione di sottoclasse, si usa la parola riservata **extends**

```
public class Dirigente extends Impiegato
{
    public String divisione;
}
```

AA 2008/09
© M.A. Alberti

4

Fondamenti di architettura e programmazione
Gerarchie di classi

Ereditarietà

- Il poter definire una struttura gerarchica tra le classi costituisce una fondamentale tecnica dell'approccio object-oriented
- facilita il disegno del software e ne promuove il ri-uso
 - Derivare nuove classi da già definite
 - Creare una gerarchia tra classi
 - Controllare l'ereditarietà con i modificatori **private** vs **protected**
 - Polimorfismo tramite ereditarietà

AA 2008/09
© M.A. Alberti

5

Fondamenti di architettura e programmazione
Gerarchie di classi

Ereditarietà

- La classe esistente viene detta **classe antenato** o **superclasse** o anche **classe base**
- La nuova classe derivata viene detta **classe figlio** o **sottoclasse**.
 - Come i nomi suggeriscono, il figlio eredita alcune caratteristiche dal genitore o antenato
 - La **sottoclasse** eredita metodi e dati dalla **superclasse**
- Due discendenti della stessa superclasse si dicono **fratelli**

AA 2008/09
© M.A. Alberti

6

Fondamenti di architettura e programmazione
Gerarchie di classi

Definire sottoclassi

- L'ereditarietà è appropriata ogni volta che esiste una relazione di tipo **è_un** (o **is_a**) tra due oggetti.
 - Il dirigente **è_un** particolare impiegato
 - Quindi la classe **Dirigente** è definita come una **sottoclasse** di **Impiegato**
 - La classe **Dirigente** eredita dalla classe **Impiegato** alcune caratteristiche e funzionalità

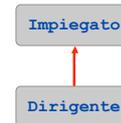
AA 2008/09
© M.A. Alberti

7

Fondamenti di architettura e programmazione
Gerarchie di classi

Definire sottoclassi

- Nel **diagramma di classe** si mette in evidenza la relazione tra classe e sottoclasse introdotta dall'ereditarietà
 - la freccia è rivolta verso la superclasse
 - L'ereditarietà implementa una relazione **is-a**, cioè il figlio **è una** versione più specifica del padre



AA 2008/09
© M.A. Alberti

8

Fondamenti di architettura e programmazione
Gerarchie di classi

Definire sottoclassi

- Un quadrato è un particolare rettangolo, che è un tipo di poligono
 - La classe **Quadrato** estende la classe **Rettangolo**, sottoclasse di **Poligono**.
- Un taxi è una particolare macchina che è un particolare veicolo
 - `class Taxi extends Macchina { ... }`
 - A sua volta la classe **Macchina** **è_un** particolare **Veicolo**
`class Macchina extends Veicolo { ... }`
- Un **Dizionario** è un tipo di **Libro**
 - [Libro.java](#), [Dizionario.java](#) con [TestDizionario.java](#)

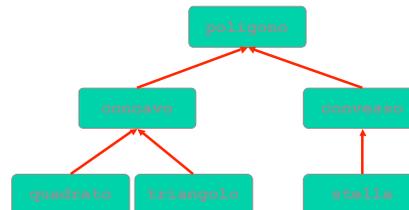
AA 2008/09
© M.A. Alberti

9

Fondamenti di architettura e programmazione
Gerarchie di classi

Gerarchie di classi

- Una sottoclasse può essere superclasse di un proprio discendente
 - Così si forma una **gerarchia di classi**



AA 2008/09
© M.A. Alberti

10

Fondamenti di architettura e programmazione
Gerarchie di classi

Disegnare gerarchie di classi

- Un buon disegno di relazioni tra classi mette tutte le caratteristiche comuni il più alto possibile in gerarchia
- Un membro di classe ereditato viene passato verso il basso nella linea di discendenza
- La gerarchia spesso deve essere estesa e modificata per soddisfare esigenze che cambiano
- Non esiste **LA** gerarchia appropriata a tutte le situazioni

AA 2008/09
© M.A. Alberti

11

Fondamenti di architettura e programmazione
Gerarchie di classi

Ereditare dalla superclasse

- La sottoclasse eredita dalla superclasse e da tutti gli antenati. In particolare eredita:
 - Tutti i membri (campi e metodi) della superclasse **accessibili** a quella sottoclasse, a meno che esplicitamente nasconda i campi o ne sovrascriva i metodi, ridefinendoli.
 - I **costruttori** non sono membri di una classe e perciò **non vengono ereditati** dalle sottoclassi.

AA 2008/09
© M.A. Alberti

12

Fondamenti di architettura e programmazione
Gerarchie di classi

Costruire un oggetto della sottoclasse

- La struttura di un oggetto della sottoclasse è come quella del padre a cui si aggiungono i campi propri.
- All'atto della creazione di un'istanza di una classe in gerarchia, viene dapprima costruito un oggetto padre quindi si inizializzano i campi del figlio.
- La chiamata al costruttore del padre può essere implicita nel caso in cui esista un costruttore senza parametri nella classe padre.
- Altrimenti viene fatta esplicitamente tramite il riferimento **super**

AA 2008/09
© M.A. Alberti

13

Fondamenti di architettura e programmazione
Gerarchie di classi

```
public class Impiegato {
    String nome;
    public Impiegato() {
        nome = "innominato";
    }
    public Impiegato(String n) {
        nome = n;
    }
}

public class Dirigente extends Impiegato {
    String divisione;
    public Dirigente(String s, String d) {
        // chiamata implicita a Impiegato()
        // la variabile nome ereditata è inizializzata a "innominato"
        divisione = d;
        nome = s;
    }
}
```

AA 2008/09
© M.A. Alberti

14

Fondamenti di architettura e programmazione
Gerarchie di classi

```
public class Impiegato {
    String nome;
    public Impiegato() {
        nome = "innominato";
    }
    public Impiegato(String n) {
        nome = n;
    }
}

public class Dirigente extends Impiegato {
    String divisione;
    public Dirigente(String s, String d) {
        super(s);
        // chiamata esplicita al costruttore della superclasse con parametri
        divisione = d;
    }
}
```

AA 2008/09
© M.A. Alberti

15

Fondamenti di architettura e programmazione
Gerarchie di classi

I membri accessibili

- Una sottoclasse eredita i membri (metodi e campi) **accessibili** della superclasse. Questi sono dichiarati:
 - **public** o **protected** nella superclasse
 - senza modificatore, purchè siano nello stesso pacchetto della superclasse
- Una sottoclasse **non** eredita i membri dichiarati
 - **private** dalla superclasse
 - con lo stesso identificatore nella sottoclasse
 - Variabili oscure
 - Metodi sovrascritti

AA 2008/09
© M.A. Alberti

16

Fondamenti di architettura e programmazione
Gerarchie di classi

Campi oscurati

```
class Sopra {
    Number unNumero;
}

class Sotto extends Sopra {
    Float unNumero;
}
```

- Il campo **unNumero** nella classe **Sotto** oscura la visibilità del campo **unNumero** della classe **Sopra**

AA 2008/09
© M.A. Alberti

17

Fondamenti di architettura e programmazione
Gerarchie di classi

I modificatori di visibilità

- Servono quindi per controllare l'ereditarietà
 - Determinano quali membri della classe vengono o non vengono ereditati
- Si ereditano i campi e i metodi dichiarati con visibilità **public**, ma non quelli con visibilità **private**
 - I campi **public** violano la regola di incapsulamento dei dati
 - Quando si vuole stabilire una relazione di ereditarietà possiamo usare un terzo modificatore di visibilità: **protected**

AA 2008/09
© M.A. Alberti

18

Fondamenti di architettura e programmazione
Gerarchie di classi

Il modificatore protected

- Il modificatore di visibilità **protected** consente ai figli di ereditare i membri di una classe base
- Il modificatore **protected** favorisce l'incapsulamento più del modificatore **public**
- Ma il modificatore **protected** non contribuisce all'incapsulamento in modo stretto come **private**

AA 2008/09
© M.A. Alberti

19

Fondamenti di architettura e programmazione
Gerarchie di classi

Il riferimento super

- Il riferimento **super** viene usato per riferirsi alla superclasse.
 - Per accedere a campi, metodi o a costruttori della superclasse
 - Spesso occorre usare il costruttore dell'antenato per inizializzare i campi di un oggetto figlio, relativi all'antenato, cioè ereditati dalla superclasse

AA 2008/09
© M.A. Alberti

20

Fondamenti di architettura e programmazione
Gerarchie di classi

Il riferimento super

- Tramite il riferimento **super** si può accedere a variabili oscurate dalla sottoclasse
 - La variabile **unNumero** della classe **Sopra** è accessibile anche dalla classe **Sotto**
`super.unNumero`
- O ai metodi sovrascritti della superclasse

AA 2008/09
© M.A. Alberti

21

Fondamenti di architettura e programmazione
Gerarchie di classi

Sovrascrivere metodi

- Una classe figlio può **sovrascrivere** la definizione di un metodo ereditato per specializzarlo
 - Un figlio può dover modificare un metodo ereditato
- Il nuovo metodo deve avere **lo stesso prototipo** del metodo della superclasse, ma un diverso corpo
- Il tipo dell'oggetto a cui è inviato il metodo determina la versione del metodo invocato

AA 2008/09
© M.A. Alberti

22

Fondamenti di architettura e programmazione
Gerarchie di classi

```
public class Impiegato {
    String nome;
    int salario;
    public String toString() {
        return "Nome: " + nome + "\n" +
            "Salario: " + salario;
    }
}

public class Dirigente extends Impiegato {
    String divisione;
    public String toString() {
        return super.toString() + "\n" +
            "Dirigente della divisione " + divisione;
    }
}
```

AA 2008/09
© M.A. Alberti

23

Fondamenti di architettura e programmazione
Gerarchie di classi

Regole per sovrascrivere metodi

- Il prototipo dei due metodi deve essere identico: stesso tipo di rientro e parametri
- Il metodo sovrascritto non può essere meno accessibile del metodo originale
- Un metodo sovrascritto non può sollevare differenti tipi di eccezioni rispetto al metodo originale

AA 2008/09
© M.A. Alberti

24

Fondamenti di architettura e programmazione
Gerarchie di classi

Sovrascrivere metodi

- Il metodo della superclasse può essere esplicitamente invocato mediante il riferimento **super**
- Se un metodo è dichiarato anche con il modificatore **final** allora non può essere sovrascritto
- La sovrascrittura può essere applicata ai campi
 - si dice che i campi sono **oscurati**
- [Massima.java](#) e la sottoclasse [Consiglio.java](#) con il driver [Messaggi.java](#)
- [Disinfettante.java](#), e la sottoclasse [Detergente.java](#)