Programmazione II

A.A. 2022-23

Prof. Maria Tortorella

Inheritance (Ereditarietà)

- Ereditarietà come meccanismo di estensione del comportamento

Ereditarietà

Aggiungere le responsabilità ad una classe

Esempi:

- Aggiungere funzionalità di colori ad una finestra di editing
- Aggiungere capacità di ordinamento alla classe ArrayList
- Aggiungere un middle name alla classe Name

Modifichiamo la classe esistente?

- Potrebbe non essere desiderabile
  - La classe è già rigorosamente verificata e robusta
  - Allargare la classe comporta una complessità aggiuntiva
  - Il codice sorgente potrebbe non essere disponibile
  - Le modifiche possono non essere consigliabili (ad esempio per le classi Java predefinite)

Ereditarietà

- Estende una classe per formare una nuova classe

```
class NewClass extends ExistingClass {

}
```

- Senza scrivere alcuna linea, NewClass eredita completamente il comportamento di ExistingClass !

Sottoclasse e superclasse

- La nuova classe consiste di:
  - Tutti i metodi della classe esistente
  - Tutte le variabili di istanza della classe esistente
  - Ogni nuovo metodo introdotto dalla nuova classe
  - Ogni nuova variabile di istanza introdotta dalla nuova classe
- La nuova classe è chiamata *sottoclasse*
- La classe esistente è chiamata *superclasse*

Un esempio: La classe Name

- Incontrata nelle lezioni precedenti
- La classe Name modella il nome di una persona

- Comportamento:
  - Ottenere le iniziali come oggetto String
  - Ottenere il nome come oggetto String, nell'ordine nome, cognome
  - Ottenere il nome come oggetto String, nell'ordine titolo, cognome, nome
  - Aggiungere o sostituire un titolo (come Signore, Sig.na, Sig)

La classe Name

```
class Name {
  public Name(String first, String last) {
    firstName = first;
    lastName = last;
    title = "";
  }

  public String getInitials() {
    String s;
    s = firstName.substring(0,1).concat(".");
    s =s.concat(lastName.substring(0,1)).concat(".");
    return s;
  }
```

La classe Name

```
  public String getLastFirst() {
      return lastName.concat(",").concat(firstName);
  }

  public String getFirstLast() {
      return title.concat(" ").
      concat(firstName).concat(" ").
      concat(lastName);
  }

  public void setTitle(String newTitle) {
      title = newTitle;
  }

  private String title, firstName, lastName;
}
```

Estendere la classe Name

- Si vuole definire una nuova classe che permetta di gestire anche il secondo nome
- Comportamento della nuova classe
  - constructor
  - get middle initial
  - 'formal name'
    - title + first name + middle initial + last name
  - È necessario mantenere il middle name come parte dello stato

Aggiungere lo stato

- L'ereditarietà permette di introdurre variabili d'istanza e metodi.

```
class ExtendedName extends Name {
...
  private String middleName;
}
```

Costruttore

È necessario inizializzare sia le variabili d'istanza dichiarate nella classe Name che quella dichiarata in ExtendedName:

- Invocazione del costruttore della superclasse: super

```
public ExtendedName(String firstName,
            String middleName, String lastName) {
      super( firstName, lastName);
      this. middleName = middleName;
}

public ExtendedName(String firstName,String lastName){
      super(firstName, lastName);
      this.middleName = "";
}
```

Il metodo getMiddleInitial

```
public String getMiddleInitial() {
      if (! middleName.equals("")){
            return middleName.substring( 0, 1);
      } else{
            return "NMN";
      }
}
```

"NMN" . . . veramente una buona scelta ??

Il metodo getFormalName

```
public String getFormalName() {
      return title + " " + firstName + " " +
                           middleName + " " + lastName;
}
```

- **Problema**: il metodo non funziona poiché le variabili title, firstName e lastName sono private alla classe Name
  - Bisogna passarle attraverso l'interfaccia pubblica della superclasse …

```
public String getFormalName() {
      return getTitle() + " " + getFirstName() + " " +
                  getMiddleName() + " " + getLastName();
}
```

Ereditarietà come relazione *is-a*

- Un oggetto della sottoclasse è anche un oggetto della superclasse
  - Esso possiede tutti i metodi e lo stato della superclasse
- L'oggetto della sottoclasse può così essere usato dovunque possa essere usato un oggetto della superclasse

```
ExtendedName exName =
      new ExtendedName(" John"," Quincy"," Adams");
Name name;
name = exName;   // OK, an ExtendedName is-a Name
```

Esempi

- Invece il contrario non è consentito

```
ExtendedName exName;
Name name = new Name(" John", "Brown");
exName = name;   // Compiler error!!
```

- Riferimenti ad oggetti Name non possono essere assegnati a variabili reference di tipo ExtendedName
  - Gli oggetti Name non sono necessariamente oggetti ExtendedName!!

Overriding di metodi

- C'è una discrepanza nel comportamento del metodo getInitials quando applicato all'oggetto ExtendedName
  - Esso non include anche l'iniziale del middleName
- Necessità di ridefinire quel metodo nella sottoclasse
- Il nuovo metodo sovrascrive (*overrides*) il metodo originale della classe Name

```
class ExtendedName extends Name {
   ...
   public String getInitials() {...}
   ...
}
```

Notare che la segnatura del nuovo (overriding) metodo deve essere identica (match) a quella del metodo originale (overridden)

Invocazione di metodi *overridden*

- Quale metodo getInitials viene eseguito in questo codice?

```
Name name = new Name("Paolo", "Rossi");
System.out.print(name.getInitials());
```

- Il metodo getInitials invocato è quello della classe Name

Invocazione di metodi *overridden*

- E in quest'altro caso?

```
ExtendedName exName =
      new ExtendedName(" Paolo", "Rossi");
System.out.print(exName.getInitials());
```

viene invocato il metodo getInitials della classe ExtendedName

Invocazione di metodi *overridden*

- E cosa succede in questo codice ?

```
Name name = new ExtendedName("Paolo", "M", "Rossi");
System.out.print(name.getInitials());
```

- La variabile reference (name) è di classe Name, mentre l'oggetto attuale (il cui riferimento è restituito dall'invocazione del costruttore ExtendedName) è di classe ExtendedName
- Quale metodo getInitials è invocato ?

Static e dynamic binding

- Name è il tipo *statico* (compile-time) della variabile reference
- Ma il tipo *dinamico* (execution-time) dell'oggetto a cui esso fa riferimento è Extended-Name !

Quando si invocano metodi *overridden*

- conta il tipo dell'oggetto attuale (tipo dinamico),
- non il tipo della variabile reference (tipo statico).

Il metodo invocato è quello della classe dell'oggetto non quello della classe della variabile reference

Polimorfismo

- Quindi, nel codice

```
Name name = new ExtendedName("Paolo", "M", "Rossi");
System.out.print(name.getInitials());
```

è il metodo getInitials di ExtendedName ad essere invocato

Questo è un altro caso di Polimorfismo dopo quello già esaminato con le Interface

Overriding vs Overloading

- Overloading
  - Stesso nome per due metodi distinti
  - Static binding (method resolution)

```
System.out.println(i);
System.out.println();
```

- Overriding
  - Permette la ridefinizione di metodi nella sottoclasse
  - Due metodi con la stessa segnatura, implementati uno nella sottoclasse ed una nella superclasse
  - Dynamic binding

```
name.getInitials()
```

Esempio

- L'ereditarietà estende una classe aggiungendo metodi e variabili di stato
- Esempio:
  Savings account = bank account con un tasso di interesse

```
class SavingsAccount extends BankAccount
{
    new methods
    new instance fields
}
```

Tra le variabili d'istanza ci dovrà avere una variabile che memorizza il tasso di interesse, `interestRate`

Esempio

- `SavingsAccount` eredita i metodi e le variabili di BankAccount

```
SavingsAccount collegeFund = new SavingsAccount(10);
// Savings account with 10% interest
collegeFund.deposit(500);
// OK to use BankAccount method with SavingsAccount object
```