Wrappers e Autoboxing

Come inserisco interi in una Collection?

```
Collection<int> coll=new ArrayList();
```

unexpected type
required: reference
found: int

(Alt-Enter shows hints)

```
g=0;
tring s:a) {
teger z;
```

Devo usare un "Wrapper": class Integer

Class Float per i float,
Class Boolean per i boolean,
Class Double per i double,
Class Char per i char...

Using wrappers

```
public class Wrappers {
    Collection<Integer> coll=new ArrayList();
    public static void main(String a[]){
        new Wrappers(a);
    }
```

```
public Wrappers(String[] a){
        for (String s:a) {
            int i=0;
            try { i=Integer.parseInt(s) }
            catch (NumberFormatException e) {
                System.out.println(s+ "non e' un intero");
                System.exit(1);
            }
            coll.add(new Integer(i));

        }
        for (Integer k:coll) {
            int x=k.intValue()+3;

            System.out.println(x);
        }; } }
```

Autoboxing and Unboxing

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

Here is the simplest example of autoboxing:

```
Character ch = 'a';
```

Using autoboxing

```
public class Wrappers {
    Collection<Integer> coll=new ArrayList();
    public static void main(String a[]){
        new Wrappers(a);
    }
public Wrappers(String[] a){
        for (String s:a) {
            int i=0;
            try { i=Integer.parseInt(s) }
            catch (NumberFormatException e) {
                System.out.println(s+ "non e' un intero");
                System.exit(1);
            }
            //coll.add(new Integer(i));
            coll.add(i);
        }
        for (Integer k:coll) {
            //int x=k.intValue()+3;
            int x=k+3;
            System.out.println(x);
        }; } }
```

Javadoc

Creazione della documentazione per un progetto Java (JavaDoc) in NetBeans

Quando viene creato, un nuovo progetto, il codice generato contiene già lo scheletro di base del JavaDoc :

```
package throwaway2;
/**
 * Qui va messa una breve descrizione della classe
 * @author ronchet
 */
public class ThrowAway2 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

Creazione della documentazione per un progetto Java (JavaDoc) in NetBeans

Occorre aggiungere una breve descrizione della classe (riga verde): se si va a capo Netbeans provvede a tenere l'allineamento e a mettere la * di inizio riga:

```
package throwaway2;
/**
 * Qui va messa la documentazione della classe,
 * che può essere
 * multilinea
 * @author ronchet
 */
public class ThrowAway2 {
    …
    }
}
```

Descrizione della classe

Occorre aggiungere una breve descrizione della classe (riga verde): se si va a capo Netbeans provvede a tenere l'allineamento e a mettere la * di inizio riga:

```
package throwaway2;
/**
 * Qui va messa la documentazione della classe,
 * che può essere
 * multilinea
 * @author ronchet
 */
public class ThrowAway2 {
    …
    }
}
```

Cos'altro documentare?

Si dovrebbero poi passare a commentare tutti i metodi e tutte le variabili di istanza.

In realtà, nella documentazione prodotta saranno di default mostrati solo metodi e variabili pubbliche, quindi ALMENO quelli devono essere documentati.

E' buona prassi documentare con Javadoc anche quelli non pubblici, almeno se non sono ovvi.

Ricordare che UN COMMENTO SBAGLIATO O NON AGGIORNATO è peggio di un commento assente!

Documentazione di metodi

Se ora apriamo una riga prima della dichiarazione del metodo, digitiamo /** (apertura del commento) e diamo return, Netbeans prepara lo scheletro di documentazione del metodo stesso (in rosso).

A noi resta da scrivere la spiegazione di cosa fa il metodo, e il significato dei parametri (testo in verde)

```
/**
* Un metodo che non fa nulla
* @param k sceglie come fare l'azione richiesta:
* se k=0 non fa niente, se k!=0 non fa nulla.
*/
void f(int k) {
   //do nothing
}
```

Documentazione delle variabili di istanza

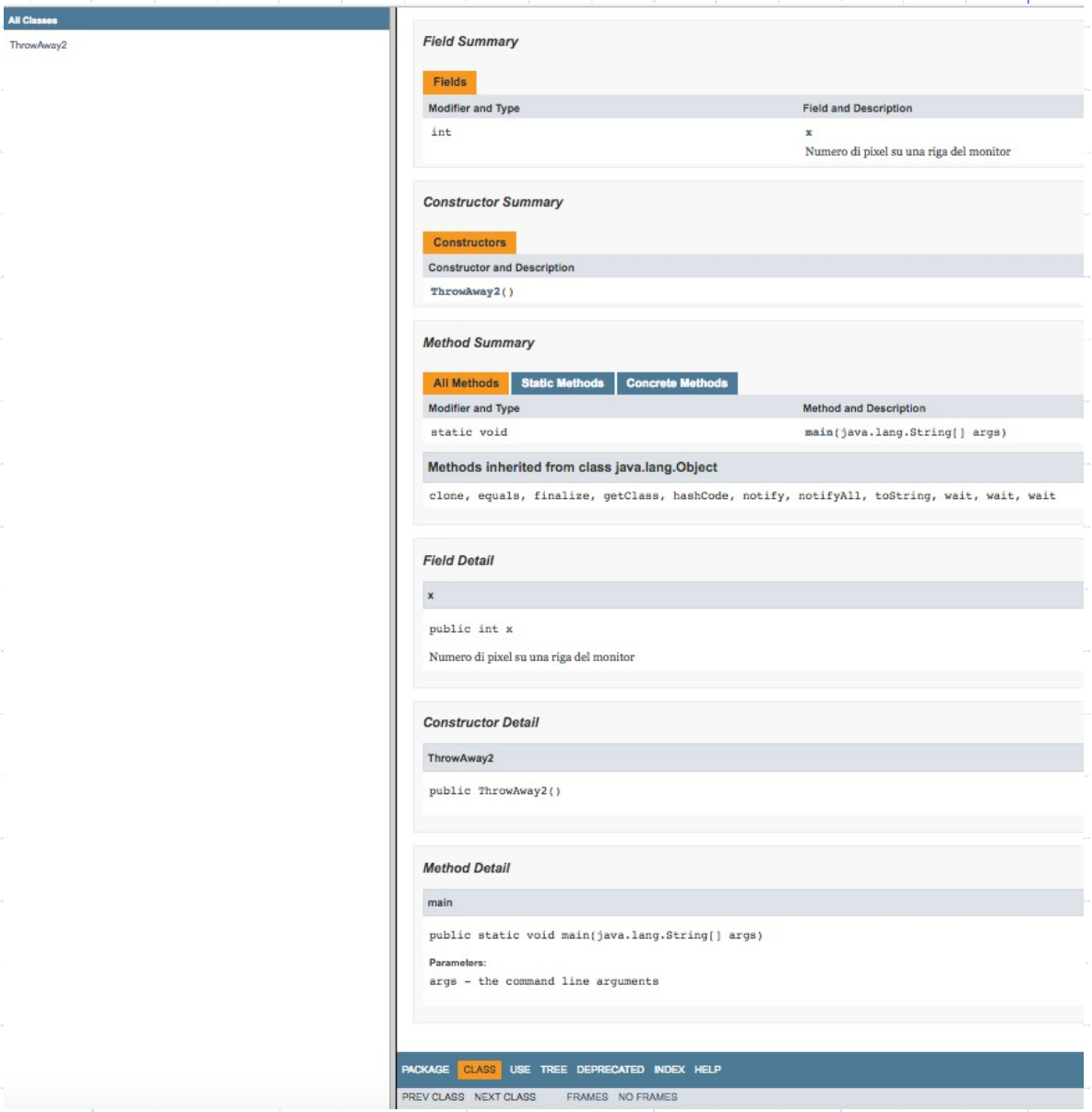
Allo stesso modo dovremo commentare il ruolo delle variabili di istanza pubbliche:

```
/**
* Numero di pixel su una riga del monitor
*/
public int x;
```

Generazione della documentazione

A questo punto la nostra classe è interamente commentata.

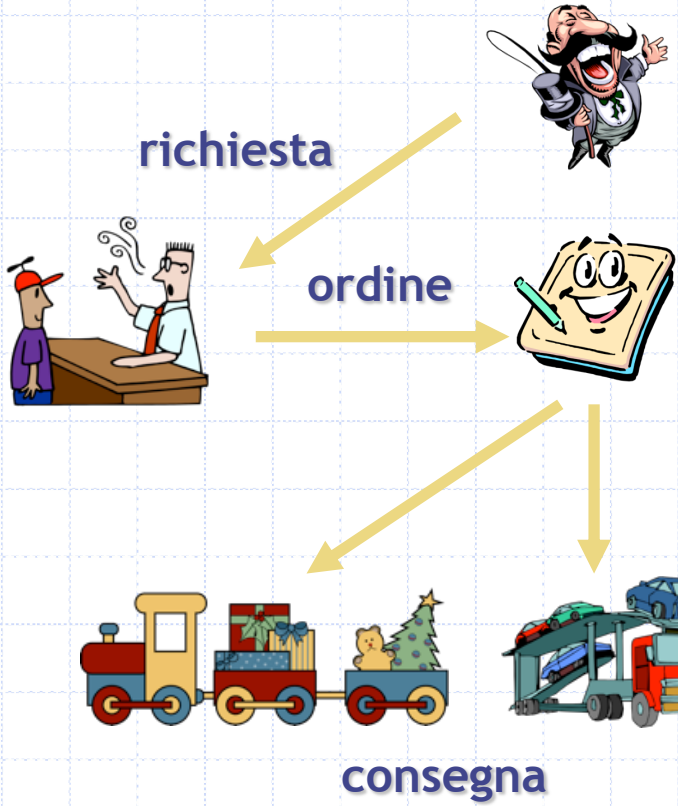
Possiamo andare nel menu Run e scegliere la voce “Generate Javadoc”. La documentazione viene generata e mostrata in una pagina web.



Unified Modeling Language (UML): una breve introduzione

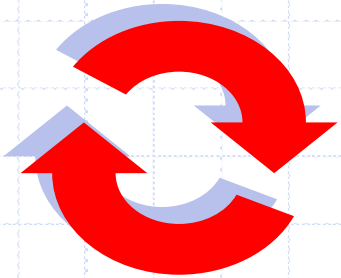
slides adattate da Gian Pietro Picco

Modellazione visuale



"Un modello cattura le parti essenziali di un sistema"

James Rumbaugh



Business Process

Computer System

Perché UML

- È il linguaggio visuale standard (OMG) per definire, progettare, realizzare e documentare i sistemi software ad oggetti
- Riunisce molte proposte esistenti (Booch, Rumbaugh e Jacobson)
- Copre l'intero processo di produzione
- È sponsorizzato dalle maggiori industrie produttrici di software

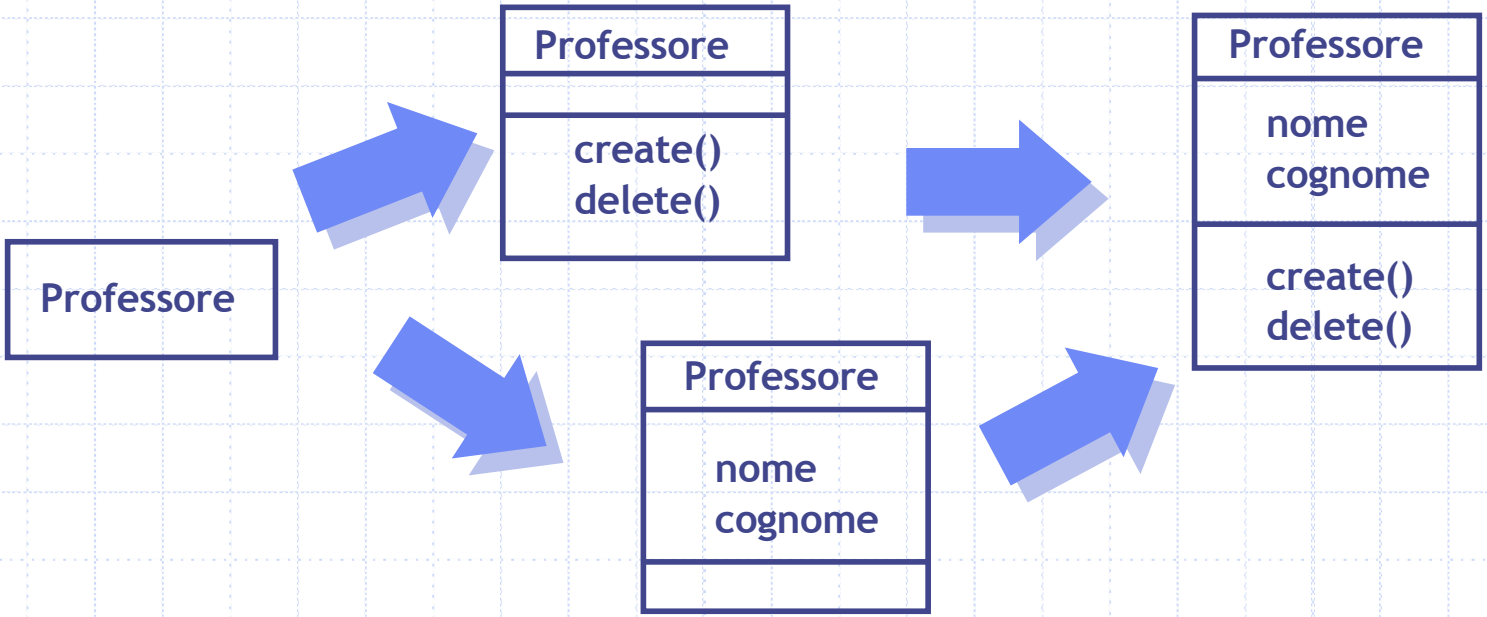
Perché UML

- Riunisce aspetti dell'ingegneria del software, delle basi di dati e della progettazione di sistemi
- È indipendente da qualsiasi linguaggio di programmazione
- È utilizzabile in domini applicativi diversi e per progetti di diverse dimensioni
- È estendibile per modellare meglio le diverse realtà applicative

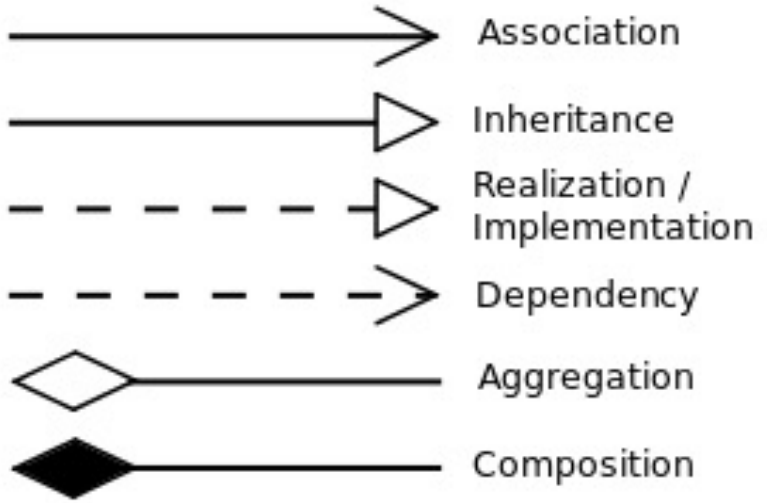
Classe

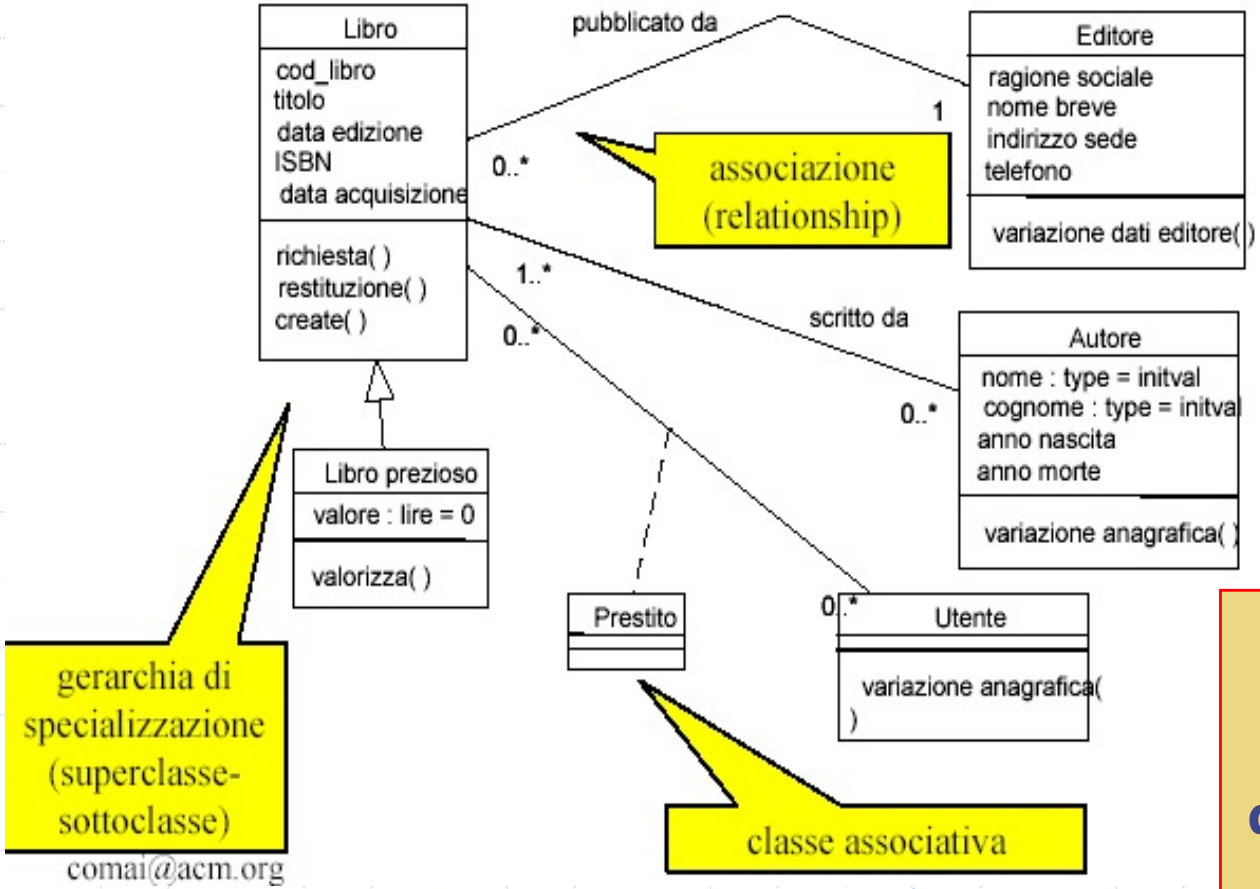
In UML è composta da tre parti

- nome
- attributi (lo stato)
- metodi (il comportamento)



Relazioni





Disegno ripreso da: Adriano Comai http://www.analisi-disegno.com/a_comai/corsi/sk_uml.htm

Sequence diagram

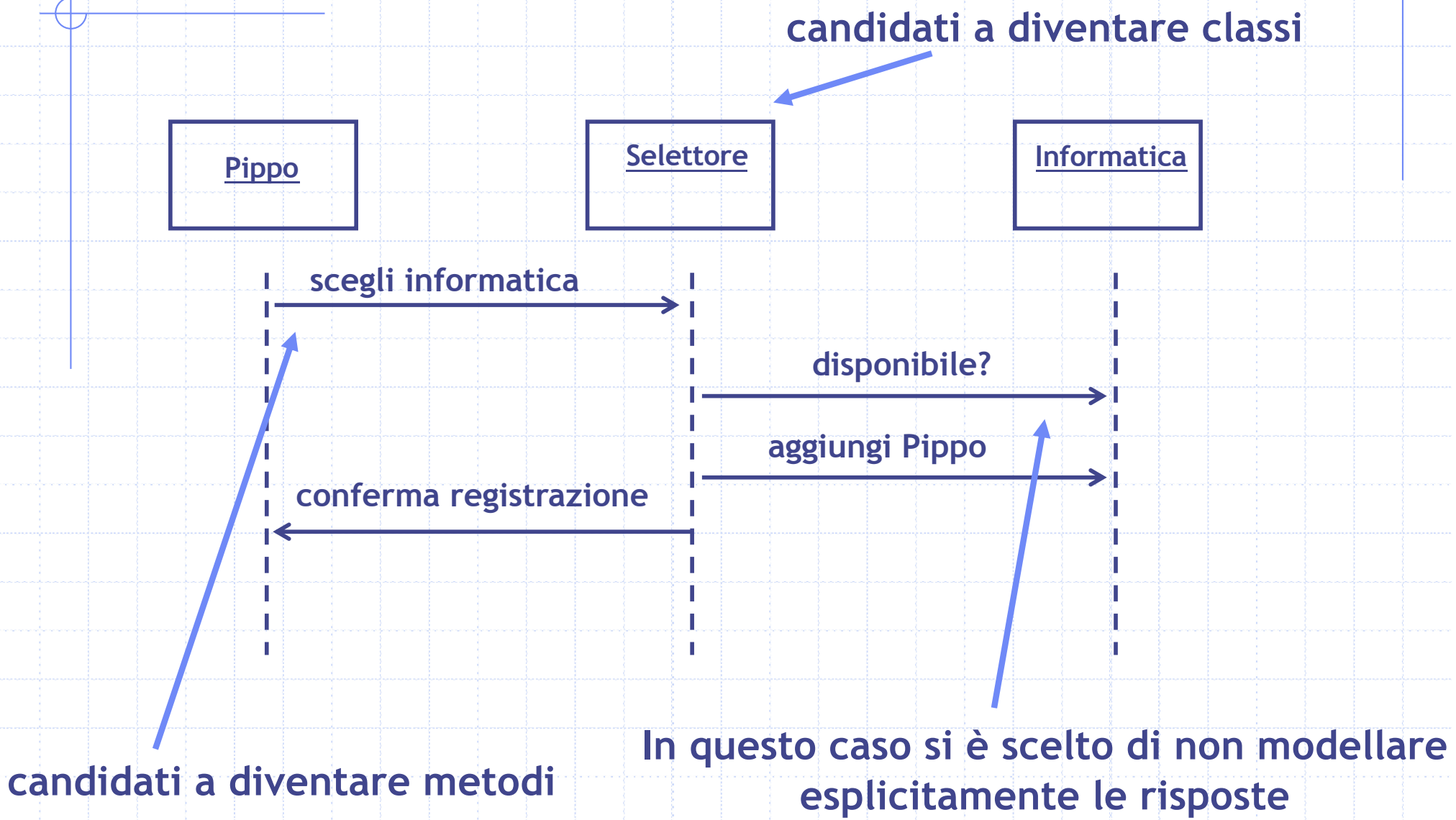
Evidenziano la sequenza temporale delle azioni

Non si vedono le associazioni tra oggetti

Usabili in due forme diverse

- generica: tutte le sequenze (esecuzioni) possibili
- istanza: una sequenza particolare, consistente con quella generica

Esempio (istanza)



Esempio
(raffinamento diagramma precedente)