SAPIENZA Università di Roma, Facoltà di Ingegneria - Sede di Rieti
Laurea in Ingegneria Informatica

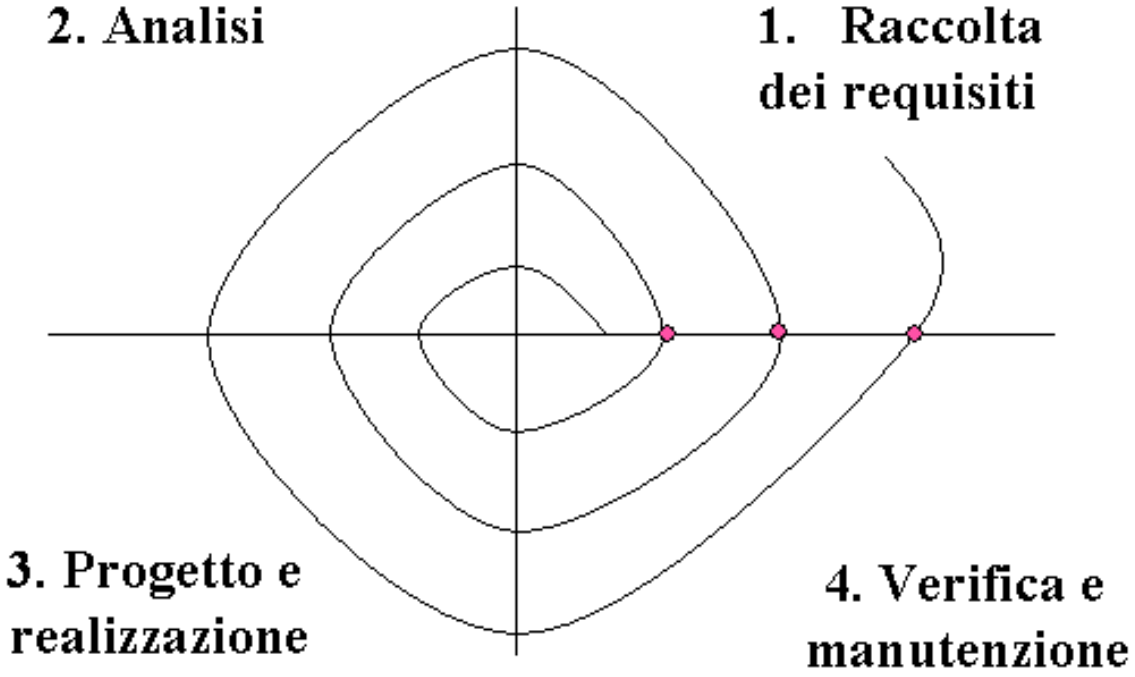
Corso di
PROGETTAZIONE DEL SOFTWARE

Prof. Emiliano Casalicchio
A.A. 2009/10

LA FASE DI PROGETTO

Fasi del ciclo di vita del software (riassunto)

1. Raccolta dei requisiti
2. Analisi
3. Progetto e realizzazione
4. Verifica e manutenzione



Corso di "Progettazione del Software": fasi 2 e 3

Progetto e realizzazione (riassunto)

Si occupa del *come* l'applicazione dovr realizzare le sue funzioni:

1. definire l'architettura del software/programma,
2. scegliere le strutture di rappresentazione,
3. produrre la documentazione (relativa a architettura, strutture dati/classi, codice),
4. scrivere il codice (del programma).

Progetto: 1, 2 e 3; realizzazione: 3 e 4

Progetto: generalitá

- In inglese: *design*.

- Alcune decisioni prese in questa fase dipendono dal linguaggio di programmazione scelto.
  - In questo corso, usiamo Java.
  - Le considerazioni che facciamo valgono in larga misura ($\geq$ 90%) per altri linguaggi orientati agli oggetti, ad es., C++, C#, VisualBasic.
  - comunque possibile definire una metodologia anche per linguaggi non orientati agli oggetti, ad es., C, Pascal.

Input alla fase di progetto

l'output della fase di analisi, ed costituito da:

- lo **schema concettuale**, formato da:
  - diagramma degli use case, delle attivitá e di sequenza,
  - diagramma delle classi e degli oggetti,
  - diagramma degli stati e delle transizioni,
- la **specifica**:
  - una specifica per ogni use case,
  - una specifica per ogni classe.

Output della fase di progetto

l'input della fase di realizzazione, ed costituito da:

1. progetto di **algoritmi** per le operazioni degli use case e delle classi UML;
2. scelta delle classi UML che hanno **responsabilit** sulle associazioni;
3. scelta/progetto delle **strutture di dati**;
4. scelta della corrispondenza fra **tipi** UML e Java;

Output della fase di progetto (cont.)

5. scelta della tecnica di gestione delle **precondizioni**;

6. scelta della **gestione delle propriet** di una classe UML;

7. scelta della rappresentazione degli **stati** in Java;

8. progetto della **API** delle principali classi Java.

Considereremo ognuno di questi elementi singolarmente.

Caso di studio

- Vedremo la fase di progetto attraverso lo studio di un caso.

- Per tale caso, considereremo gi fatta la fase di analisi, per cui, oltre ai requisiti, assumeremo di avere come input:
  - lo schema concettuale,
  - la specifica delle classi e degli use case.

Caso di studio: scuola elementare

Requisiti. L'applicazione da progettare riguarda le informazioni su provveditorati scolastici, scuole elementari e lavoratori scolastici. Di ogni scuola elementare interessa il nome, l'indirizzo e il provveditorato di appartenenza. Di ogni provveditorato interessa il nome e il codice attribuitogli dal Ministero. Dei lavoratori scolastici interessa la scuola elementare di cui sono dipendenti, il nome, il cognome e l'anno di vincita del concorso.

Esistono solamente tre categorie di lavoratori scolastici, che sono fra loro disgiunte: dirigenti, amministrativi e insegnanti. Dei primi interessa il tipo di laurea che hanno conseguito, dei secondi il livello (intero compreso fra 1 e 8), mentre dei terzi interessano le classi in cui insegnano, e, per ogni classe, da quale anno.

Caso di studio: scuola elementare (cont.)

Ogni insegnante insegna in almeno una classe, e ogni classe ha almeno due insegnanti. Di ogni classe interessa il nome (ad es. "IV A") e il numero di alunni.

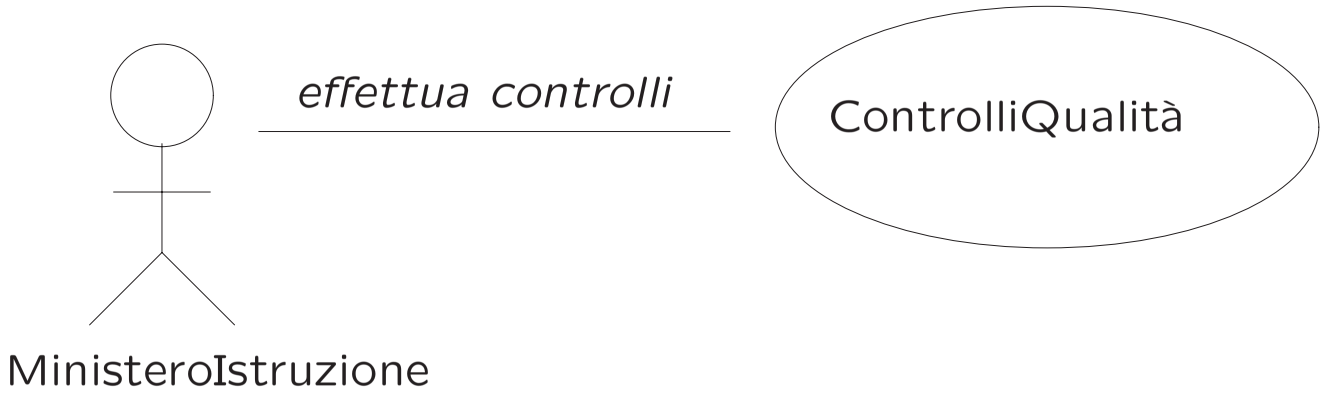
Ogni lavoratore scolastico pu essere in servizio oppure assente, in particolare per ferie, malattia o aspettativa. Dall'aspettativa o dalla malattia si pu tornare solamente in servizio, mentre se se si in ferie si pu andare anche in malattia.

Caso di studio: scuola elementare (cont.)

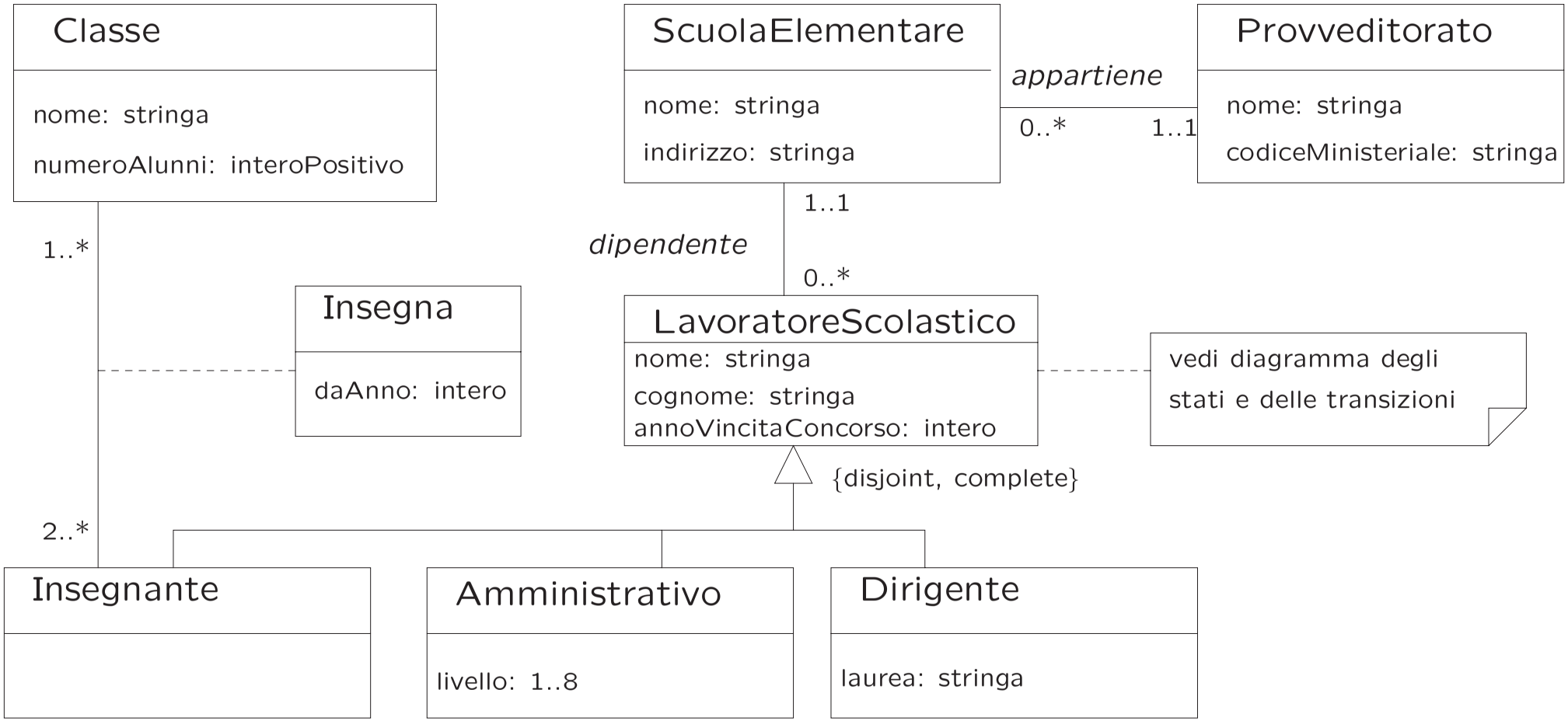
Il Ministero dell'Istruzione deve poter effettuare, come cliente della nostra applicazione, dei controlli sull'insegnamento. A questo scopo, si faccia riferimento ad uno use case che prevede le seguenti operazioni:

- data una classe e l'anno corrente, calcolare quanti insegnanti della classe hanno vinto il concorso da pi di 15 anni;
- dato un insegnante, calcolare il numero totale di alunni a cui insegna;
- dato un insieme di insegnanti, calcolare il numero medio di alunni a cui insegnano.

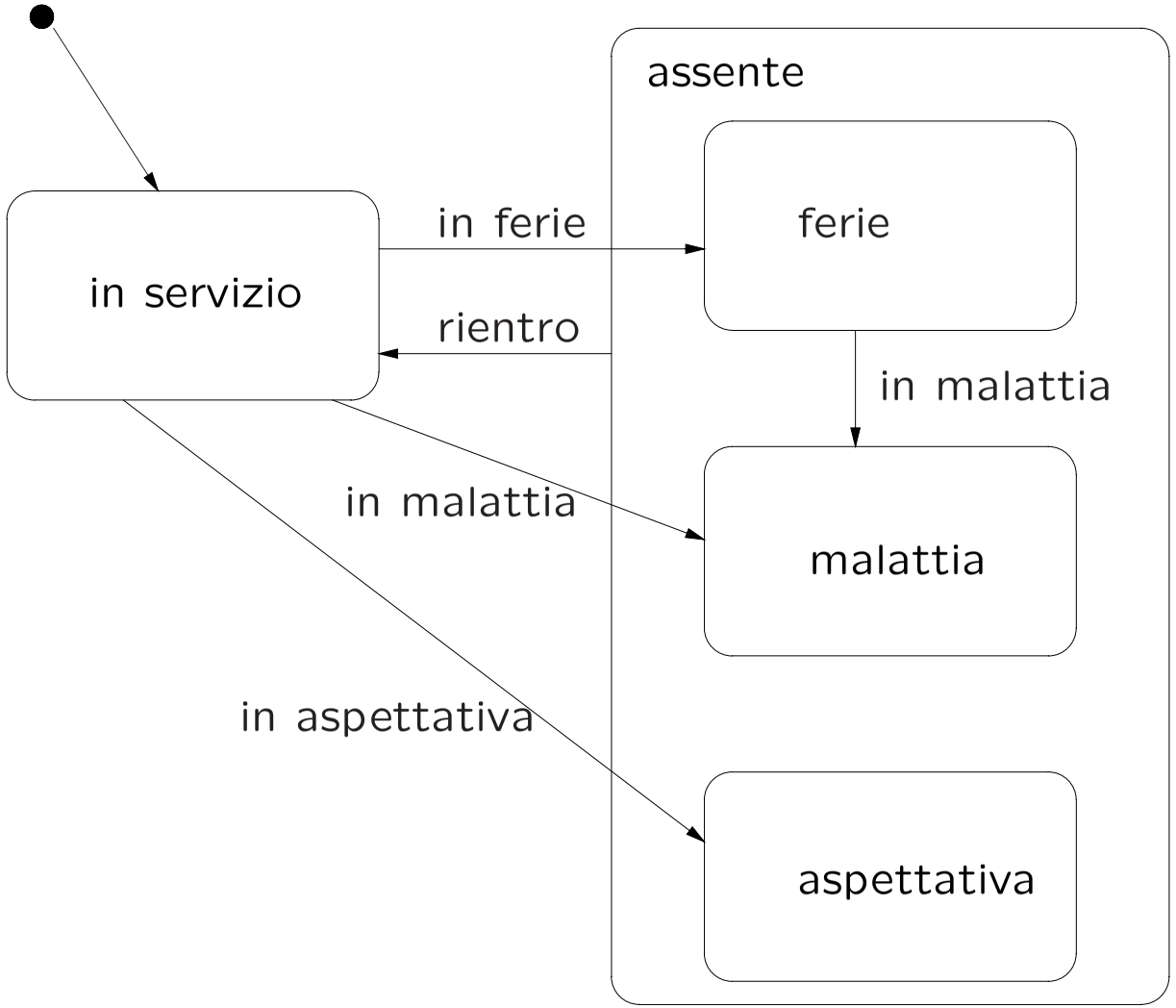
Caso di studio: diagramma degli use case



Caso di studio: diagramma delle classi UML



Caso di studio: diagramma degli stati e delle transizioni classe LavoratoreScolastico



Caso di studio: specifica dello use case

InizioSpecificaUseCase ControlliQualit

NumeroInsegnantiDaAggiornare *(c: Classe, a: intero): intero*

pre: nessuna

post: *result* il numero di insegnanti di *c* la cui data di vincita del concorso inferiore o uguale ad *a* - 15

NumeroAlunniPerDocente *(i: Insegnante): intero*

pre: nessuna

post: *result* il numero totale di alunni a cui *i* insegna

NumeroMedioAlunniPerDocente *(ins: Insieme(Insegnante)): reale*

pre: c' almeno un insegnante in *ins*

post: *result* il numero medio di alunni a cui gli insegnanti di *ins* insegnano

FineSpecifica

Caso di studio: specifica formale dello use case

InizioSpecificaUseCase ControlliQualit

NumeroInsegnantiDaAggiornare *(c: Classe, a: intero): intero*

pre: true

post: definiamo Ins come l'insieme

$\{i | i \in Insegnante \wedge \langle i, c\rangle \in Insegna \wedge i.annoVincitaConcorso \leq a - 15\}$.

result = card(Ins)

NumeroAlunniPerDocente *(i: Insegnante): intero*

pre: true

post: definiamo Cla come l'insieme $\{c | c \in Classe \wedge \langle i, c\rangle \in Insegna\}$.

result = $\sum_{c \in Cla} c.numeroAlunni$

...

Caso di studio: specifica formale (cont.)

...

NumeroMedioAlunniPerDocente *(ins: Insieme(Insegnante)): reale*

pre: $ins \neq \emptyset$

post: $result = \frac{\sum_{i \in ins} NumeroAlunniPerDocente(i)}{card(ins)}$

FineSpecifica

Progetto: algoritmi

- L'obiettivo fornire alla fase di realizzazione un algoritmo:
  - per ogni operazione della specifica di ogni use case,
  - per ogni operazione della specifica di ognuna delle classi UML.

- Possiamo specificare gli algoritmi in una delle maniere tradizionali:
  - attraverso *pseudocodice* ;
  - attraverso *diagrammi di attivitá*.

Progetto: algoritmi (cont.)

- Nella specifica di un algoritmo:
  - assumiamo che ne siano verificate le precondizioni (ritorneremo in seguito su questo aspetto);
  - diamo per scontate operazioni su insiemi (ad es., unione, intersezione);
  - prescindiamo dalla rappresentazione delle strutture di dati.

- Il progetto di algoritmi fornisce un criterio di **verifica** del diagramma delle classi: se non riusciamo a scrivere l'algoritmo perch non abbiamo rappresentato nel diagramma alcune informazioni necessarie, allora dobbiamo **ritornare alla fase di analisi**.

Caso di studio: algoritmi

- Per l'operazione **NumeroInsegnantiDaAggiornare** adottiamo il seguente algoritmo:

```
int result = 0;
per ogni link l di tipo insegna in cui c  coinvolto
    Insegnante i = l.insegnante;
    se (a - i.annoVincitaConcorso) > 15 allora
        result++;
return result;
```

- Per l'operazione **NumeroAlunniPerDocente** adottiamo il seguente algoritmo:

```
int result = 0;
per ogni link l di tipo insegna in cui i  coinvolto
    Classe c = l.Classe;
    result += c.numeroAlunni;
return result;
```

Caso di studio: algoritmi (cont.)

- Per l'operazione **NumeroMedioAlunniPerDocente** adottiamo il seguente algoritmo:

```
int tot_alunni = 0;
per ogni insegnante i di ins;
    per ogni link l di tipo insegna in cui i  coinvolto
        Classe c = l.Classe;
        tot_alunni += c.numeroAlunni;
float result = tot_alunni / cardinalit(ins);
return result;
```

o, in alternativa

```
int tot_alunni = 0;
per ogni insegnante i di ins;
    tot_alunni += NumeroAlunniPerDocente(i);
float result = tot_alunni / cardinalit(ins);
return result;
```

Esercizio 1

- Considerare un ulteriore use case che prevede le seguenti operazioni:
  - dato un dirigente, calcolare quante classi ci sono nella sua scuola;
  - dato un provveditorato, verificare se vero che tutte le scuole elementari di sua appartenenza hanno esattamente un dirigente;
  - dato un provveditorato p ed un intero m, determinare quante sono le scuole elementari di appartenenza di p che hanno almeno una classe con pi di m alunni.

- Produrre la specifica dello use case e dell'algoritmo (descrizione verbale e activity diagram) per ognuna delle sue operazioni.

Progetto: responsabilit sulle associazioni

Prima di realizzare una classe UML che coinvolta in un'associazione, ci dobbiamo chiedere se la classe ha **responsabilit** sull'associazione.

Diciamo che **una classe C ha responsabilit sull'associazione A**, quando, per ogni oggetto x che istanza di C vogliamo poter eseguire opportune operazioni sulle istanze di A a cui x partecipa, che hanno lo scopo di:

- **conoscere** l'istanza (o le istanze) di A alle quali x partecipa,
- **aggiungere** una nuova istanza di A alla quale x partecipa,
- **cancellare** una istanza di A alla quale x partecipa,
- **aggiornare** il valore di un attributo di una istanza di A alla quale x partecipa.

Progetto: responsabilit sulle associazioni (cont.)

Per ogni associazione A, **deve** esserci almeno una delle classi coinvolte che ha responsabilit su A.

I criteri per comprendere se una classe C ha responsabilit sull'associazione A sono i seguenti:

1. esiste una parte (ad es., una frase) nel documento dei requisiti, da cui si evince che per ogni oggetto x che istanza di C vogliamo poter eseguire almeno una delle operazioni di "conoscere, aggiungere, cancellare, aggiornare";

2. esiste un'operazione in uno use case per la quale non possibile realizzare il corrispondente algoritmo senza che la classe C abbia tale responsabilit;

3. la responsabilit logicamente implicata dai vincoli di molteplicit delle associazioni.

Caso di studio: responsabilit

Prendiamo in considerazione il **criterio 1**.

- *Di ogni scuola elementare interessa [. . .] il provveditorato di appartenenza*.
  → *ScuolaElementare* ha responsabilit su *appartiene*.

- *Dei lavoratori scolastici interessa la scuola elementare di cui sono dipendenti*.
  → *LavoratoreScolastico* ha responsabilit su *dipendente*.

- *[Degli insegnanti] interessano le classi in cui insegnano*.
  → *Insegnante* ha responsabilit su *insegna*.

Caso di studio: responsabilit (cont.)

Prendiamo in considerazione il **criterio 2**.

- Prendiamo in considerazione l'algoritmo dell'operazione **NumeroInsegnantiDaAggiornare**. evidente che per la sua realizzazione necessario che, a partire da un oggetto *c* che istanza di *Classe* possiamo conoscere le istanze di *insegna* alle quali *c* partecipa.
  $\rightarrow$ *Classe* ha responsabilit su *insegna*.

- Prendiamo in considerazione gli algoritmi delle operazioni **NumeroAlunniPerDocente** e **NumeroMedioAlunniPerDocente**. evidente che per la loro realizzazione necessario che, a partire da un oggetto *i* che istanza di *Insegnante* possiamo conoscere le istanze di *insegna* alle quali *i* partecipa.
  $\rightarrow$ *Insegnante* ha responsabilit su *insegna*.

  Si noti che eravamo gi a conoscenza di questa responsabilit.

Molteplicitá e responsabilit'a

- L'esistenza di alcuni vincoli di molteplicit di associazione ha come conseguenza l'esistenza di responsabilit su tali associazioni.

- In particolare, quando una classe *C* partecipa ad un'associazione *A* con un vincolo di:

  1. molteplicit massima finita, oppure

  2. molteplicit minima diversa da zero,

  la classe *C* **ha necessariamente responsabilit su** *A*.

- Il motivo, che sar ulteriormente chiarito nella fase di realizzazione, risiede nella necessit di poter interrogare gli oggetti della classe *C* sul numero di link di tipo *A* a cui partecipano, al fine di verificare il soddisfacimento del vincolo di molteplicit.

Caso di studio: responsabilitá (cont.)

Prendiamo in considerazione il **criterio 3**.

- Esiste un vincolo di molteplicitá minima diversa da zero (2..*) nell'associazione *insegna*.
  → *Classe* ha responsabilit su *insegna*.

  Si noti che eravamo gi a conoscenza di questa responsabilitá.

- Anche tutte le altre responsabilit determinate in precedenza possono essere evinte utilizzando questo criterio.

Caso di studio: tabella delle responsabilitá

Possiamo riassumere il risultato delle considerazioni precedenti nella seguente *tabella delle responsabilit*.

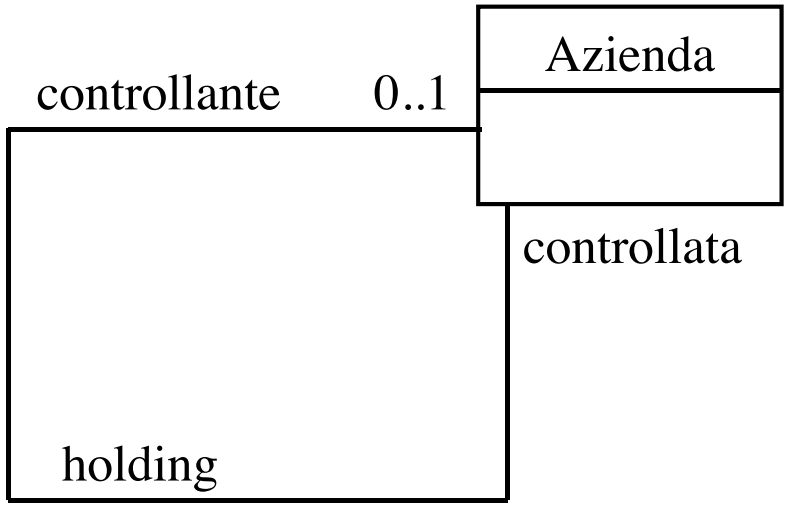
Associazione	Classe	ha resp.
insegna	*Classe*	$S^{2,3}$
	Insegnante	$S^{1,2,3}$
dipendente	*ScuolaElementare*	NO
	LavoratoreScolastico	$S^{1,3}$
appartiene	*ScuolaElementare*	$S^{1,3}$
	Provveditorato	NO

1. dai requisiti
2. dagli algoritmi
3. dai vincoli di molteplicit

Criterio di **verifica**: per ogni associazione deve esserci almeno un "S".

Responsabilit'ei ruoli

Quanto detto vale anche per il caso in cui l'associazione coinvolga pi volte la stessa classe. In questo caso il concetto di responsabilit si attribuisce **ai ruoli**, piuttosto che alle classi.



Ad esempio, la classe *Azienda* potrebbe avere la responsabilit sull'associazione *holding*, solo nel ruolo *controllata*. Questo significa che, dato un oggetto *x* della classe *Azienda*, vogliamo poter eseguire operazioni su *x* per conoscere l'eventuale azienda controllante, per aggiornare l'azienda controllante, ecc.

Esercizio 2

Si considerino lo use case e gli algoritmi forniti come soluzione all'esercizio 1.

Quali sono le responsabilit che si possono evincere da tali algoritmi?

Scelta delle strutture di dati

- Prendendo in considerazione:
  - il diagramma delle classi,
  - la tabella delle responsabilit,
  - gli argomenti delle operazioni e i loro valori restituiti;
  - gli algoritmi

  possibile determinare se si avr bisogno di **strutture** per la rappresentazione **dei dati** della nostra applicazione.

Scelta delle strutture di dati (cont.)

- Facendo riferimento allo caso di studio, notiamo che:
  - poich la classe *Insegnante* ha responsabilit sulla associazione *insegna*, la cui molteplicit 1..*, per la realizzazione di quest'ultima avremo bisogno di rappresentare *insiemi di link*;
  - lo stesso si pu dire prendendo in considerazione la classe *Classe*;
  - per rappresentare l'input dell'operazione *NumeroMedioAlunniPerDocente* avremo bisogno di un opportuno insieme di insegnanti.

Scelta delle strutture di dati (cont.)

- In generale, emerge la necessit di rappresentare *collezioni omogenee* di oggetti.

- Per fare ci, utilizzeremo il *collection framework* di Java 1.5 che, attraverso l'uso dei *generics* permette di realizzare collezioni omogenee, in particolare:

  - `Set<Elem>`, `HashSet<Elem>`, `...`,

  - `List<Elem>`, `LinkedList<Elem>`, `...`,

Corrispondenza fra tipi UML e Java

- Prendendo in considerazione:
  - il diagramma delle classi,
  - la specifica degli use case e delle classi,
  - la scelta delle strutture di dati,
  - gli algoritmi in pseudocodice,

  possibile compilare una lista dei **tipi** UML per i quali dobbiamo decidere la rappresentazione in Java.

- In generale, per la rappresentazione opportuno scegliere un tipo base Java (`int`, `float`, ...) o una classe Java di libreria (`String`, ...) ogni volta ci sia una chiara corrispondenza con il tipo UML.

Caso di studio: tabella di corrispondenza dei tipi UML

Possiamo riassumere il risultato delle nostre scelte nella seguente *tabella di corrispondenza dei tipi* UML.

Tipo UML	**Rappresentazione in** Java
intero	`int`
interoPositivo	`int`
1..8	`int`
reale	`double`
stringa	`String`
Insieme	`Set`

Corrispondenza fra tipi UML e Java (cont.)

Per due casi servono ulteriori considerazioni:

1. quando il tipo UML necessario per un attributo di classe con una sua molteplicit (ad es., *NumeroTelefonico: Stringa* $\{0..*\}$);

2. quando non esiste in Java un tipo base o una classe predefinita che corrisponda chiaramente al tipo dell'attributo UML (ad es., *Indirizzo*).

Ulteriori considerazioni: caso 1

- In questo caso, coerentemente con la scelta precedente relativa alle strutture di dati, scegliamo di rappresentare l'attributo mediante una classe Java per *collezioni omogenee* di oggetti, come `Set`.

- Va notato che tali classi non consentono la rappresentazione di insiemi di valori di tipi base (`int`, `float`, etc.), ma **solamente di oggetti**.

- Di conseguenza, per la rappresentazione dei valori atomici dobbiamo utilizzare classi Java di libreria, quali `Integer`, `Float`, etc.

- Ad esempio, per rappresentare un ipotetico attributo *AnniVincitaConcorsi: intero* $\{0..*\}$, useremo un oggetto costruito mediante un'espressione come:

```
HashSet<Integer> s = new HashSet<Integer>();
```

Ulteriori considerazioni: caso 2

- Per quanto riguarda la realizzazione di tipi UML tramite classi Java, notiamo che, dal punto di vista formale, il tipo UML andrebbe specificato ulteriormente tramite le *operazioni* previste per esso.

- In questo corso, affrontiamo questo aspetto in una maniera *intuitiva e non formale*, descrivendo in linguaggio naturale le operazioni.

- L'approccio seguito simile a quello della realizzazione di *classi* UML (per maggiori dettagli rimandiamo quindi alla successiva parte del corso: "*La fase di realizzazione*"), con alcune alcune regole da seguire per le funzioni speciali:

  `toString()`: si pu prevedere di farne overriding, per avere una rappresentazione testuale dell'oggetto.

Ulteriori considerazioni: caso 2 (cont.)

`equals()`: **necessario** fare overriding della funzione `equals()` ereditata dalla classe `Object`.

Infatti due valori sono uguali solo se sono lo stesso valore, e quindi il comportamento di default della funzione `equals()` non corretto.

`hashCode()`: **necessario** fare overriding della funzione `hashCode()` ereditata dalla classe `Object`.

Infatti in Java deve sempre valere il principio secondo il quale *se due oggetti sono uguali secondo* `equals()` *allora questi devono avere lo stesso codice di hash secondo* `hashCode()`. Quindi poiché ridefiniamo `equals()` dobbiamo anche ridefinire coerentemente a detto principio `hashCode()`.

Ulteriori considerazioni: caso 2 (cont.)

`clone()`: ci sono due possibilit:

1. Nessuna funzione della classe Java effettua side-effect. In questo caso, `clone()` non si ridefinisce (gli oggetti sono *immutabili*).

2. Qualche funzione della classe Java effettua side-effect. In questo caso, poich i moduli clienti hanno tipicamente la necessit di copiare valori (ad esempio, se sono argomenti di funzioni) **si mette a disposizione la possibilit di copiare un oggetto**, rendendo disponibile la funzione `clone()` (facendo overriding della funzione `protected` ereditata da `Object`).

Realizzazione di tipi UML

- A titolo di esempio, vediamo la realizzazione del tipo UML *Data*, inteso come aggregato di un giorno, un mese ed un anno validi secondo il calendario gregoriano, e per cui le operazioni previste sono:
  - selezione del giorno, del mese e dell'anno;
  - verifica se una data sia precedente ad un'altra;
  - avanzamento di un giorno.

- Realizziamo il tipo UML *Data* mediante la classe Java `Data`, rappresentando il giorno, il mese e l'anno come campi dati `private` di tipo `int`.

- Scegliamo di realizzare l'operazione di avanzamento di un giorno mediante una funzione Java che fa side-effect sull'oggetto di invocazione.

Esempio: la classe Java Data

```
// File Tipi/Data.java
public class Data implements Cloneable {
  // SERVE LA RIDEFINIZIONE DI clone(), in quanto una funzione fa side-effect
  public Data() {
    giorno = 1;
    mese = 1;
    anno = 2000;
  }
  public Data(int a, int me, int g) {
    giorno = g;
    mese = me;
    anno = a;
    if (!valida()) {
      giorno = 1;
      mese = 1;
      anno = 2000;
    }
  }
  public int giorno() {
    return giorno;
  }
  public int mese() {
    return mese;
  }
```

```
public int anno() {
  return anno;
}
public boolean prima(Data d) {
  return ((anno < d.anno)
          ||  (anno == d.anno && mese < d.mese)
          ||  (anno == d.anno && mese == d.mese && giorno < d.giorno));
}
public void avanzaUnGiorno() {
  // FA SIDE-EFFECT SULL'OGGETTO DI INVOCAZIONE
  if (giorno == giorniDelMese())
    if (mese == 12) {
      giorno = 1;
      mese = 1;
      anno++;
    }
    else {
      giorno = 1;
      mese++;
    }
  else
    giorno++;
}
public String toString() {
  return giorno + "/" + mese + "/" + anno;
}
```

```
public Object clone() {
  try {
    Data d = (Data)super.clone();
    return d;
  } catch (CloneNotSupportedException e) {
    // non può accadere, ma va comunque gestito
    throw new InternalError(e.toString());
  }
}
public boolean equals(Object o) {
  if (o != null && getClass().equals(o.getClass())) {
    Data d = (Data)o;
    return d.giorno == giorno && d.mese == mese && d.anno == anno;
  }
  else return false;
}
public int hashCode() {
  return giorno + mese + anno; //possiamo naturalmente realizzare una
                               //funzione di hash più sofisticata
}

// CAMPI DATI
private int giorno, mese, anno;

// FUNZIONI DI SERVIZIO
private int giorniDelMese() {
```

```
    switch (mese) {
    case 2:
      if (bisestile()) return 29;
      else return 28;
    case 4: case 6: case 9: case 11: return 30;
    default: return 31;
    }
  }
  private boolean bisestile() {
    return ((anno % 4 == 0) && (anno % 100 != 0))
      || (anno % 400 == 0);
  }
  private boolean valida() {
    return anno > 0 && anno < 3000
      && mese > 0 && mese < 13
      && giorno > 0 && giorno <= giorniDelMese();
  }
}
```

Esercizio 4

possibile realizzare l'operazione di avanzamento di un giorno **senza fare side-effect** sull'oggetto di invocazione, ovvero nella cosiddetta maniera *funzionale*.

Ad esempio, ci possibile prevedendo un metodo Java pubblico con la seguente dichiarazione.

```
// File Tipi/DataFunzionale.java
public class DataFunzionale {
// ...
  public static DataFunzionale unGiornoDopo(DataFunzionale d) {
    // NON FA SIDE-EFFECT
    // ...
  }
```

Si noti che la funzione statica e riceve l'input tramite il suo argomento, e non tramite l'oggetto di invocazione.

Realizzare la funzione Java `unGiornoDopo()` in maniera che restituisca il giorno successivo al suo argomento, senza modificare quest'ultimo.

Esercizio 5

Considerare la seguente versione modificata dei requisiti:

"Dei lavoratori scolastici interessa [...] e la residenza (indirizzo, CAP, citt, provincia)."

Produrre la corrispondente versione modificata del diagramma delle classi e della tabella di corrispondenza dei tipi UML, progettando anche le eventuali classi Java che si dovessero rendere necessarie.