INFORMATICA III
Parte B - Progettazione e Algoritmi

Introduzione

Patrizia Scandurra patrizia.scandurra@unibg.it
Università degli Studi di Bergamo a.a. 2012-13

Contatti

- **Docente:** Patrizia Scandurra (parte B – Progettazione ed algoritmi)
- **Email:**{patrizia.scandurra}@unibg.it
- **Sito web (parte B)**: http://cs.unibg.it/scandurra/INF3ProgAlg2013.html
- **Ricevimento (parte B)**:
  - Edificio B, terzo piano, ufficio 3
  - Su appuntamento o dopo la lezione

Obiettivi

Fornire agli studenti

- conoscenze teoriche
- metodi e
- strumenti di sviluppo utili

per **progettare e implementare "in grande" ed "in piccolo"una applicazione software** attraverso

- un **processo di sviluppo "agile"**
- uno **sviluppo orientato alle componenti** e all'efficienza degli **algoritmi e strutture dati** impiegate
- il linguaggio di **programmazione ad oggetti Java** e l'**ambiente Eclipse**

Natura

- **Lezioni frontali e di esercitazione (parte B)**
  - si dovranno svolgere alcuni esercizi
  - entreremo nei dettagli del codice
  - si dovrà alla fine costruire una applicazione SW complessa
- **Non informativo sulle tecnologie recenti**
  - Le tecnologie cambiano rapidamente, ma i principi rimangono evolvendo
- **Prerequisiti**: fondamenti di informatica, ingegneria del software e notazione UML, programmazione Java, elementi di analisi matematica su serie e successioni.

Argomenti (parte B)

- **Design, analisi e sviluppo di codice algoritmo**:
  - complessità del calcolo e notazione asintotica
  - metodologie di progettazione di algoritmi (incrementale,divide-et-impera,greedy,programmazione dinamica)
  - realizzazione di algoritmi e tipi astratti di dati in Java
  - strutture dati e algoritmi fondamentali (liste, pile, code), algoritmi di ordinamento, alberi e loro gestione, alberi di ricerca (alberi binari di ricerca, B-tree), applicazioni degli alberi per il processamento di documenti XML, tabelle hash, grafi e loro rappresentazione e gestione, cammini minimi.
- **Processi di sviluppo agili** modellazione, sviluppo e testing con AMDD (Agile Model-driven Development)
- **Design e sviluppo di architetture SW**: component-based software engineering, service-oriented engineering, design pattern
- **Metriche e refactoring di un'applicazione SW**: definizione/calcolo di metriche, Jdepend, refactoring di applicazioni, scenari di deployment
- **Testing e copertura di una applicazione SW**: Test Driven Development (TDD) e Extreme Programming (XP), unit testing con Junit, il tool di copertura EMMA e il plug-in Eclipse ECLEMMA
- **Sviluppo di GUI e componenti grafiche in Java**: JFC/Swing e Java2D, gestione eventi, MVC e componenti Swing di base, Jigloo GUI builder, cenni SWT

Materiale didattico (parte B)

- **Lucidi e dispense** distribuite con password attraverso il sito del docente http://cs.unibg.it/scandurra/INF3ProgAlg2013.html

- **C. Demetrescu, I. Finocchi, G. F. Italiano: Algoritmi e strutture dati**, **McGraw-Hill**, ISBN: 978-88-386-6468-7, seconda edizione, Gennaio 2008 http://www.ateneonline.it/demetrescu2e/homeA.asp

- **Tutorial su strumenti e tool** scaricabili on-line dalla piattaforma di e-learning ILIAS: http://elearning2.unibg.it/ilias4/repository.php?cmd=frameset&ref_id=2094

Per approfondimenti:

- **C.A.Shaffer: A practical Introduction to data structures and algorithm analysis**. http://people.cs.vt.edu/~shaffer/Book/Java3e20100119.pdf
- **The Object Primer 3rd Edition: Agile Model Driven Development with UML 2**, Cambridge University Press, 2004 ISBN 0-521-54018-6

Modalità d'esame (parte B)

- L'esame consta di una **parte scritta, e di una discussione orale sull'applicazione software** che avete progettato e sviluppato
  - Le funzionalità di tale applicazione sono libere, ma l'idea va concordata con il docente e il SW prodotto dovrà esibire certe caratteristiche obbligatorie

- Il giorno dell'appello (prova scritta) consegnerete anche il progetto (su CD) e la documentazione stampata
  - I progetti verranno discussi nello stesso giorno o nei giorni successivi su appuntamento

Modalità di valutazione (1)

Da 0 a 10 per la **parte scritta**: 6 = sufficiente (**BARRIERA per il progetto**)

Progetto SW e voto finale: da 0 a 10 punti per ognuna delle seguenti caratteristiche "pesate":

- ***Risultati prova scritta*** (di cui si è raggiunta almeno la sufficienza) **15%**
- ***Interesse del problema affrontato e dimensione del progetto*** **10%**
  - Premio l'originalità, difficoltà, l'interdisciplinarietà, fondamenti teorici, l'interesse potenziale di altri, dimensione del progetto e delle librerie, ecc.
  - L'idea va comunque comunicata al docente!
- ***Bontà delle soluzione proposta:*** **45%**
  - *Architettura (15%)*: opportuna divisione in package (anche tramite analisi statica con strumenti come jdepend, ad esempio presenza di cicli tra i packages), design pattern, stile di programmazione, warning del javac, ecc..
  - Algoritmi e strutture dati (15%) impiegate
  - *Validazione (15%)*: risultati ottenuti ad esempio dal testing e dal piano di test, grado di automazione del testing (che sia ripetibile), dalla copertura con Emma, ecc..

Modalità di valutazione (2)

- ***Progettazione, sviluppo e documentazione*** **20%**
  - *Valutazione della modellazione agile*: uso di UML per la modellazione dei casi d'uso (requisiti funzionali), per il design dell'architettura (diagramma delle componenti e di deployment), diagramma delle classi, piani di test, uso di metodi formali (Reti di Petri, Abstract State Machines) per le parti più "critiche", ecc..
  - *Valutazione dello sviluppo agile*: uso dei metodi (testing, copertura, refactoring, ecc..), strumenti (tool e Plug-in Eclipse) e tecnologie (librerie, APIs, ecc..) spiegati a lezione e nei tutorial
  - *Valutazione della documentazione* a corredo

- ***Presenza attiva in aula e lab*** (svolgimento degli esercizi proposti) **10%**

Caratteristiche obbligatorie (1)

- **Progettazione e documentazione** (file di testo, word, html, o altro) prodotta con le tecniche/strumenti di design imparati nel corso di Ingegneria del Software, relativamente a
  - Requisiti del sistema (funzionalità ed eventualmente altri requisiti non funzionali)
  - Progettazione dell'architettura ("in grande") con sviluppo agile e UML (casi d'uso e diagramma delle componenti/deployment)
    - Uso di UML (diagrammi delle classi e altri diagrammi) e ER, Reti di Petri, Abstract State Machines eventualmente per le funzionalità più critiche
  - Progettazione "in piccolo" di algoritmi e strutture dati
  - Piano di test e spiegazione delle decisioni fatte durante l'implementazione
  - Manuale d'uso e di installazione

Caratteristiche obbligatorie (2)

- **Implementazione**
  - Costrutti OO di Java: uso di interfacce e classi astratte, famiglie di costruttori, ereditarietà, overloading e overriding di metodi, membri static, final, applicazione dei design pattern, generics (tipi o metodi generici), ecc.
  - Algoritmi: algoritmi user-defined, uso di JFC (tra cui List e HashTable)
  - Librerie esterne: ad es. log4j o altre (jcurses, jexcelapi, o di Jakarta)
  - Unit testing: sviluppare e documentare casi di test con Junit
  - Copertura: copertura del codice con Emma (o altri tool) e tale copertura deve essere documentata
  - Documentazione del codice: completamente documentato con JavaDoc
  - I/O in formato XML: con SAX o DOM (o utilizzare un parser generator come AntLR o Javacc)
  - GUI: un'interfaccia grafica (differenziare la UI con le funzionalità dell'applicazione); preferite l'uso del pattern Model View Controller

Caratteristiche obbligatorie (3)

- **Distribuzione**
  - Al di fuori dell'IDE: installazione e funzionamento stand-alone con un installer o con uno script o con Java web start. Usate fatjar se volete mettere più jar in un unico Jar.
  - In linea di principio, non deve richiedere al docente l'installazione di SW ausiliario (DBMS o altro) per farlo funzionare. Se necessario, preferite l'uso di Java DB (http://developers.sun.com/javadb/ , di soli 2MB!) e notificatelo al docente.
  - Il codice deve essere sviluppato (e consegnato al docente) in un (unico) progetto di Eclipse e deve includere librerie e risorse necessarie per compilare/eseguire i file e i casi di test.
    - I link alle risorse devono essere *relativi* in modo che spostando il progetto su vari PC esso continui a funzionare. Se utilizzate altri progetti ausiliari, anche questi vanno inclusi·

Progettare in “piccolo” ed in “grande” con una metodologia agile

Introduzione

Metodologia agile

Differenziazione di metodi e modelli di sviluppo:

- **Metodologie pesanti** per i vecchi metodi come il *Modello a cascata*
- **Metodologie iterative** per i metodi come il *Modello a spirale*
- **Metodologie agili** (o leggere) che coinvolgono quanto più possibile il committende, ottenendo in tal modo una elevata reattività alle sue richieste
  - *Agile alliance*, una organizzazione no-profit creata allo scopo di diffonderle
  - Esistono un certo numero di tali metodologie (XP, SCRUM, AMDD,ecc..)
    - Adotteremo **AMDD = versione "agile" del Model Driven Development**

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Dialogo con l’utente che espone le sue necessità, interpretazione di massima delle specifiche e primo documento di analisi.

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Definizione delle specifiche per mezzo di tecniche formali. Stesura di un documento di analisi dei requisiti da sottoporre all’utente.

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Progettazione in grande:
Si analizza la suddivisione del software per componenti funzionali e lo scambio dati tra questi ultimi.

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Progettazione in piccolo: Per ogni componente si fa un’analisi di dettaglio e si definisce l’algoritmo.

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Scelta del linguaggio di programmazione e sviluppo delle singole parti con azione di correzione degli errori (prototipi).

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle prestazioni*
   2. *Documentazione*
   3. *Manutenzione*

Di norma un team diverso da quello di progetto. A questo punto va fatta anche l’integrazione tra le parti.

Ciclo di vita del software

Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle pr*
   2. *Documentazione*
   3. *Manutenzione*

Può essere scritta anche durante lo sviluppo, man mano che si completa il software.

Ciclo di vita del software

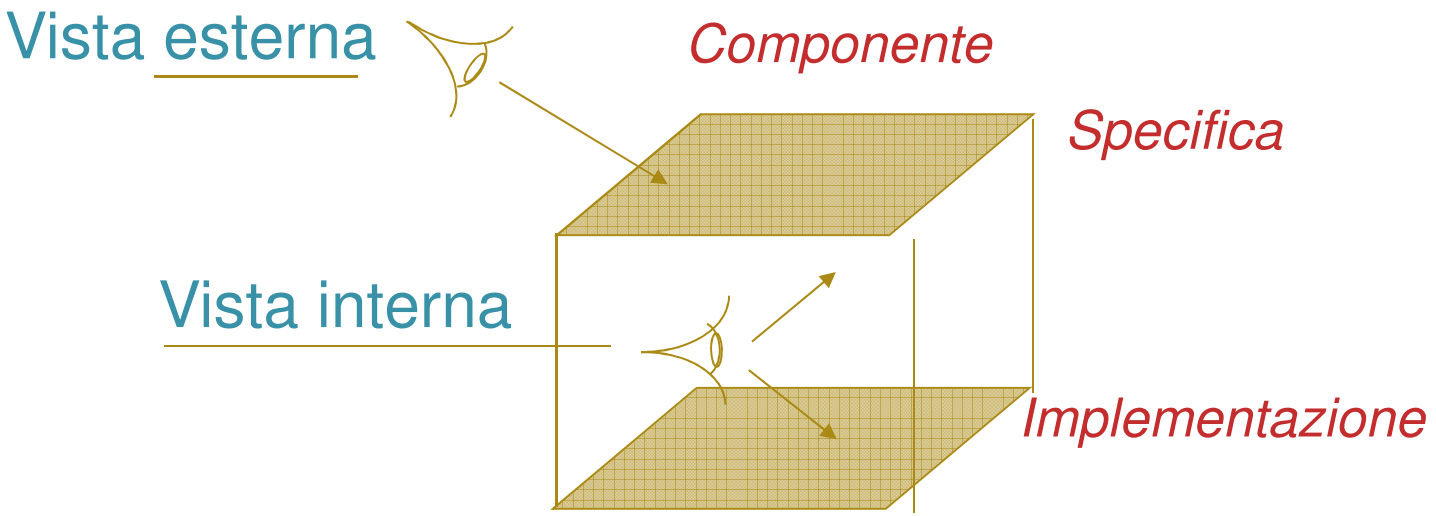
Passi “comuni” a tutte le metodologie:

1. Prima dell’implementazione
   1. *Studio di fattibilità*
   2. *Descrizione del problema*
   3. *Progetto del programma*
   4. *Scelta o sviluppo dell’algoritmo e analisi*
2. Implementazione
   1. *Codifica*
   2. *Debugging*
3. Dopo l’implementazione
   1. *Testing, verifica e valutazione delle pr*
   2. *Documentazione*
   3. *Manutenzione*

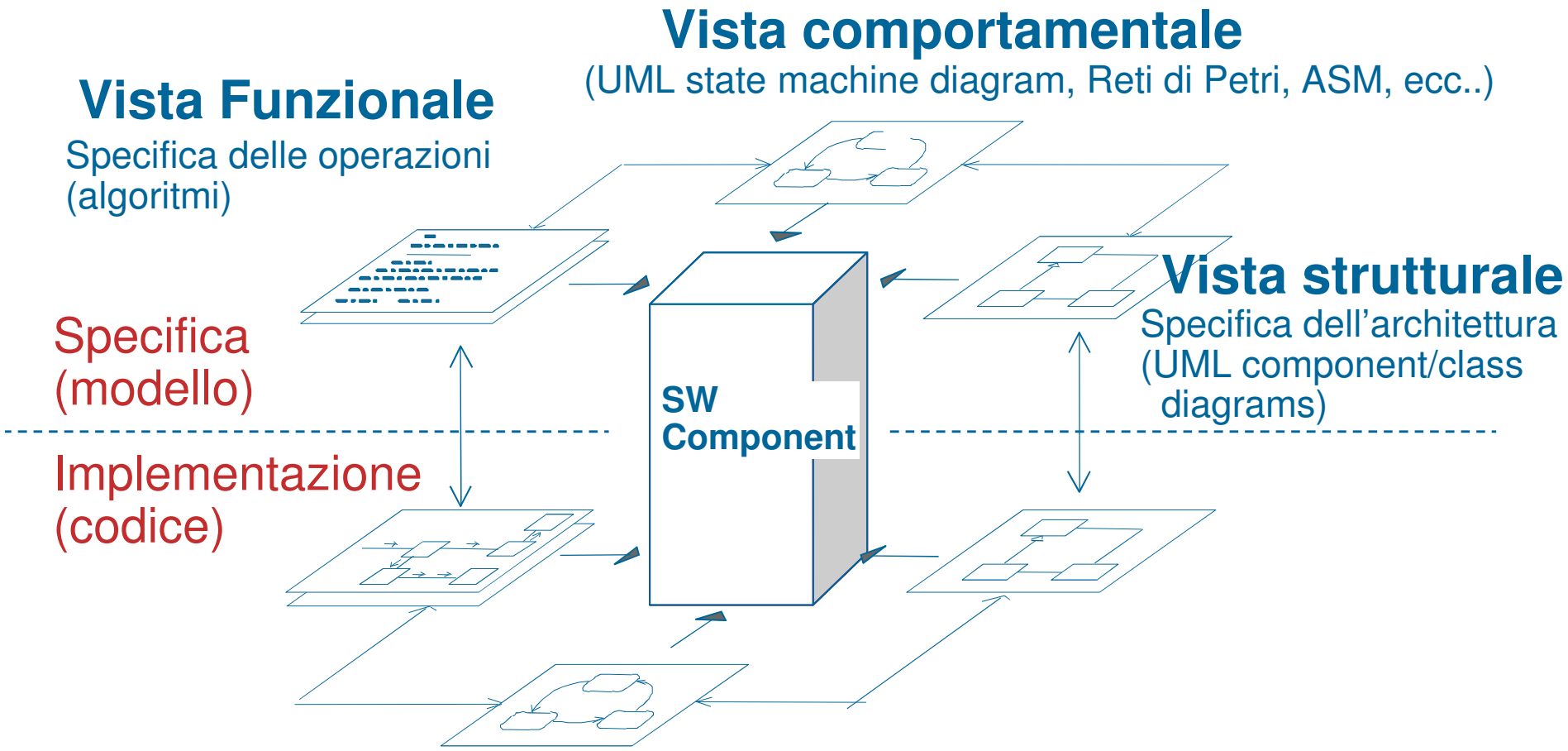
Effettuata dopo la distribuzione del prodotto, con l’aiuto degli utenti: nuove release e/o versioni.

Component-based development: principio di incapsulamento

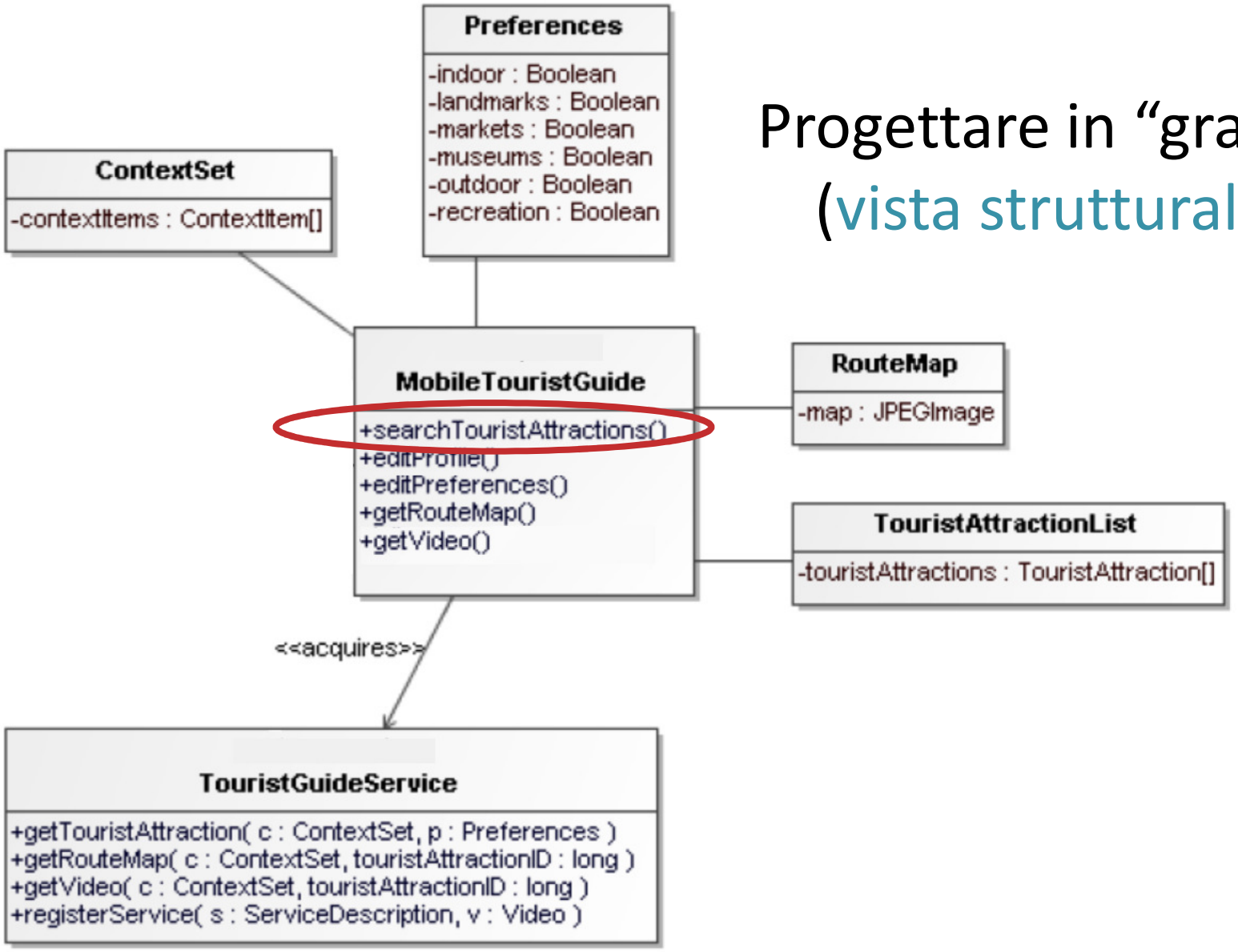
- Una netta separazione tra:
  - **La vista esterna**: **l'interfaccia esposta** dalla componente (come parte della specifica) in termini di funzioni (operazioni) fornite/richieste, struttura e comportamento
  - **La vista interna**: **dettagli implementativi** su funzionalità, struttura e comportamento interni da tenere nascosti



Ingegnerizzazione multi-view



Esempio: Mobile Tourist Guide



Progettare in “grande”
(vista strutturale)

Esempio: Mobile Tourist Guide

Progettare in “piccolo” (vista operazionale)

Name	searchTouristAttractions
Description	Searches for tourist attractions depending on the user’s preferences and the current context (e.g. location, time, weather)
Receives	-
Returns	TouristAttractionList
Sends	TouristGuideService.getTouristAttraction()
Reads	ContextSet, Preferences
Changes	TouristAttractionList
Body	- Pseudocodice algoritmo <DA DEFINIRE>
Precondition	Preferences have been set up, Context Sources are available
Postcondition	TouristAttractionList contains suitable attractions