Object oriented approach and UML

SoftEng
http://softeng.polito.it

Goals

The goals of this chapter are to

- introduce the object oriented approach to software systems development
- introduce UML notation
  - use cases
  - sequence diagrams
  - class diagrams
  - statecharts diagrams

SoftEng
http://softeng.polito.it

Summary

- The object oriented approach has been the more influential, both in research and practice, in software system development in the past 10 years.
- UML is the dominant notation based on the object oriented approach.
- This chapter presents the OO approach and part of the UML notation.

SOftEng http://softeng.polito.it

Outline

Object Oriented Approach and UML
- Approaches to modularity
- Procedural approach
- OO approach
- UML
  - Object and Class diagram
  - Use cases
  - Dynamic models
  - Physical models

SOftEng http://softeng.polito.it

Approaches to modularity

SoftEng http://softeng.polito.it

Product Principles

- P2, Divide and conquer
  - modularity
  - (high) cohesion and (low) coupling
  - information hiding

SoftEng http://softeng.polito.it

Approaches

- Given the P2 principle, how to implement it?
- Procedural approach
- Object oriented approach

SoftEng
http://softeng.polito.it

Procedural

- Procedural approach
  - module = procedure/function
  - support for analysis, design: Structured Analysis, Structured Design
  - support for coding: C, Pascal, Fortran, ..

SoftEng
http://softeng.polito.it

Object oriented

- Object oriented approach
  - module = class
  - support for analysis, design: UML
  - support for coding: C++, Java, Smalltalk, C#

SoftEng http://softeng.polito.it

Procedural approach

- module1 = procedure
- module2 = data
- relation1 = call procedure
  - w/without parameter passing, forth and back
- relation2 = rd/wr data

- coupling
  - call relation: low
  - rd relation: higher
  - wr relation: highest

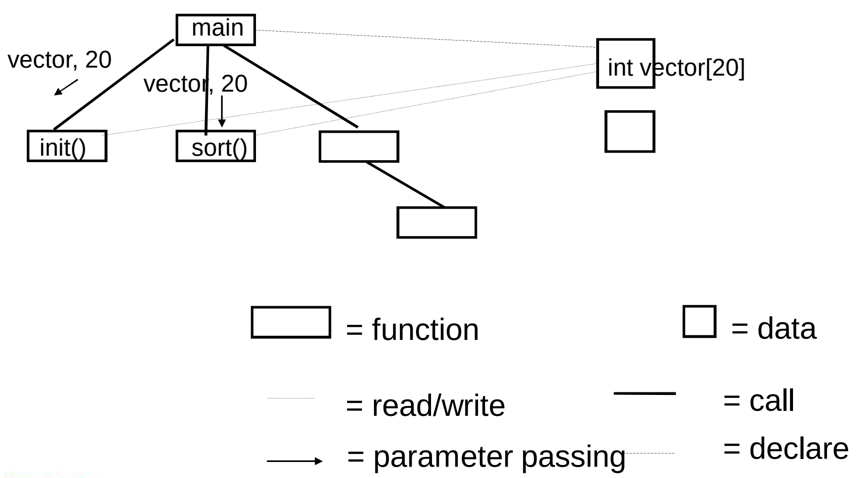
SoftEng http://softeng.polito.it

Vector – less disciplined

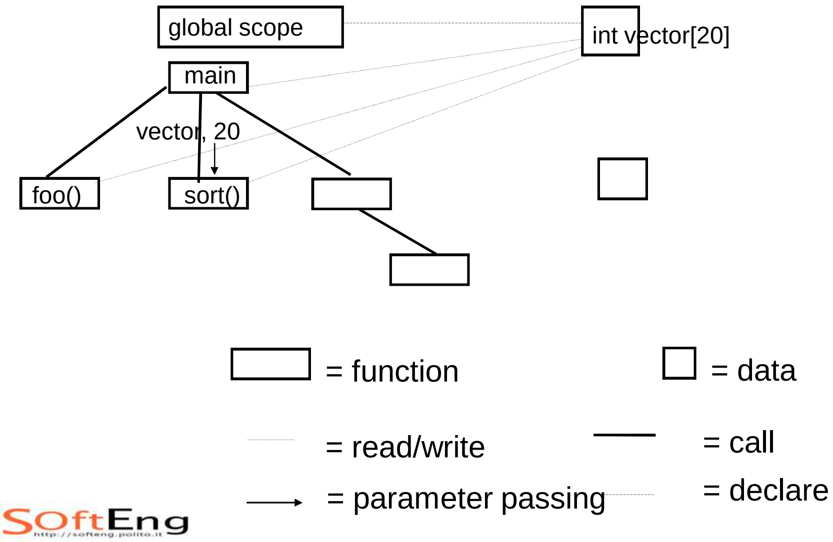
```
int vector[20];
void sort(int [] v, int size) { // sort };
void foo(){   vector[5] = 44;}
int main(){
      for (i=0; i<20; i++) {  vector[i ]=0; };
      sort(vector, 20);
      vector[4] = 33;
}
```

Rd wr can happen anywhere

Modules and relationships



Modules and relationships



SoftEng

Vector – more disciplined

```
void init (int [] v, int size) {
      for (i=0; i<size; i++) {  v[i ]=0; }};
void sort(int [] v, int size) { // sort };
int main(){
     int vector[20];
     init(vector, 20);
     sort(vector, 20);
}
```

Rd/wr only in functions that receive vector

SoftEng

Problems

- With global declaration, rd/wr relation can happen between data and any other function, without explicit declaration (parameter passing)
- if it can happen, it will happen
  - especially during maintenance/evolution
- coupling increases

SoftEng http://softeng.polito.it

- root problem is no explicit link between (structured) data and procedures working on it
  - init(), sort() and vector[20] are not linked
  - they should, as they work in symbiosis
    - parameter passing should be avoided
    - while rd/wr relationship should be confined within sort() init()
    - concept of object

SoftEng http://softeng.polito.it

OO approach – Class

```
class vector{
     private:
        int v[20];
   public:
           vector(){   // same as init }
           sort(){   // same as sort }

}
```

SOftEng http://softeng.polito.it

OO approach – object

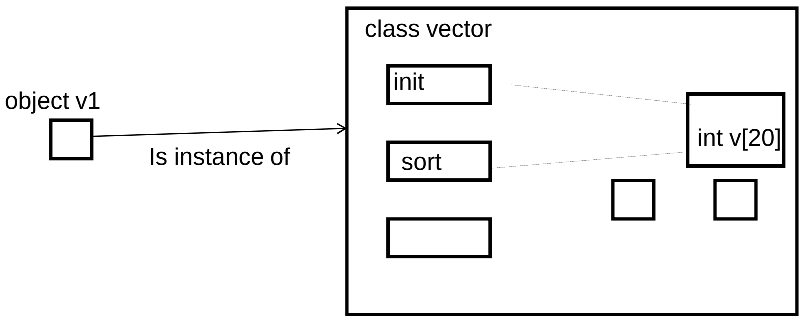
```
int main() {
   vector v1, v2;   //
   v1.sort();

}
```

SOftEng http://softeng.polito.it

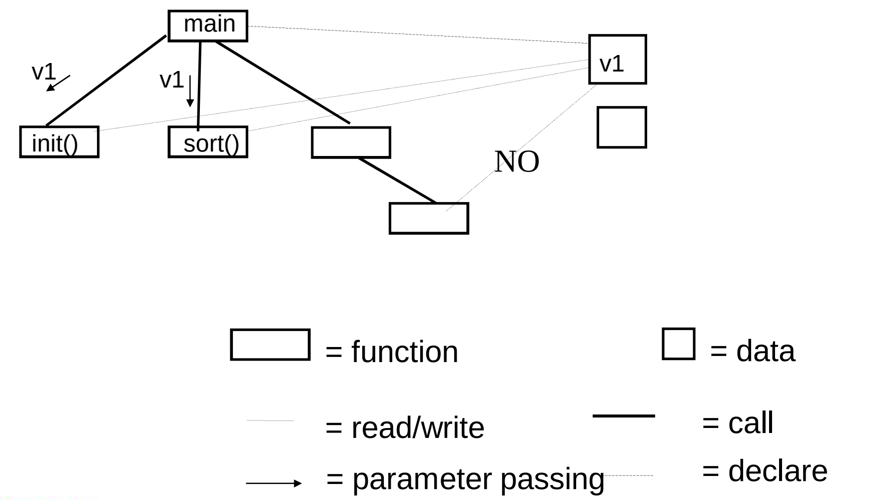
OO approach

- module1 = procedure
- module2 = data
- module3/4 = object / class
- relation1 = message passing
  - similar to procedure call with parameter passing
- relation2 = rd/wr data
- coupling
  - call relation: low
  - rd relation: higher
  - wr relation: highest



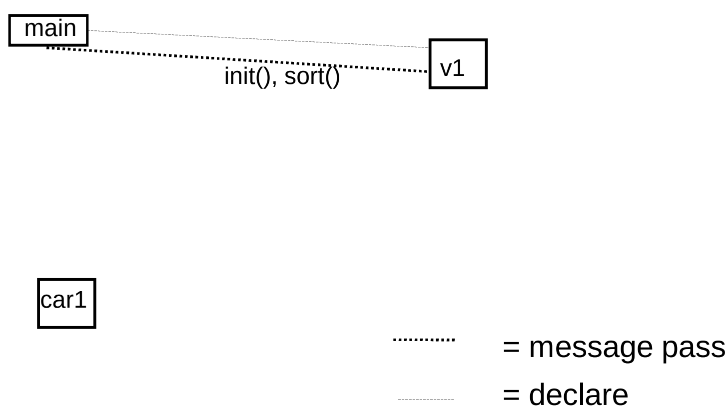
- class describes structured data and procedures that can rd/wr them
- object v1 is instance of (is described by) class
- no rd/wr outside class

Modules and relationships



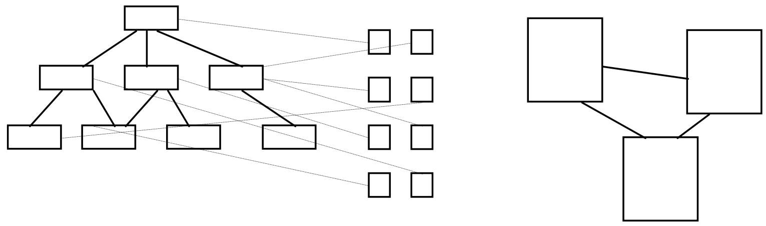
SoftEng

More OO



SoftEng

Results



– In oo world objects exchange messages
– coupling between objects is lower
  – message passing vs. procedure call
    – objects hide r/w relationship
  – less relationships among objects
    – objects are higher level of abstraction
– more complex systems can be built

SOftEng http://softeng.polito.it

Message passing vs. procedure call

- Message passing
- Control mechanism
  - same
- Data exchange
  - reference to object is passed
  - receiver can send messages, cannot rd/wr object

- Procedure call
- Control mechanism
  - same
- Data exchange
  - object is passed
  - receiver can rd/wr object

SOftEng http://softeng.polito.it

Message passing vs. procedure call

- Message passing

```
void foo(vector v){
   v.sort(); //
   v.[14] = 7; // NO
}

int main(){
  vector v1;
  vector v2;
  foo(v1);   foo(v2);
}
```

- Procedure call

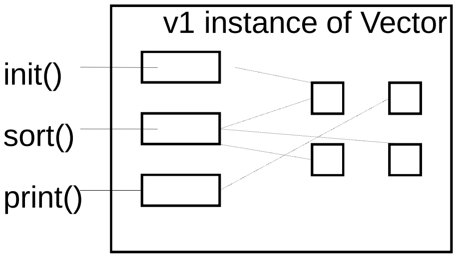
```
void foo(int vector[]){

 vector[14] = 7; //
}

int main(){
  int v1[20];
  foo(v1);
}
```

Interface

- set of messages an object can answer to

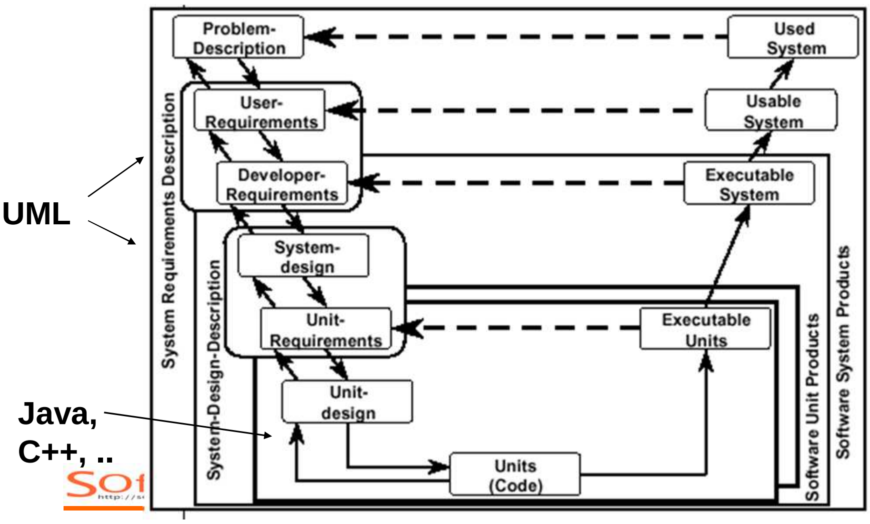


P2 revised

- objects / classes are better modularization elements
  - by construction message passing has (much) lower coupling than procedure call and rd/wr
- designer has to decide 'right' classes to implement information hiding

SoftEng

OO and process



UML

UML

- Unified Modeling Language
- standardized by OMG, Object Management Group

- Resources
  - www.cetus-links.org
  - Fowler, UML Distilled, 3rd edition, Addison Wesley

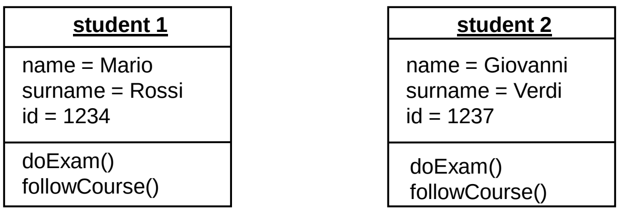
Modeling dimensions vs. UML diagrams

- Structure, entities, concepts
  - Class diagram
  - Package diagram, component diagram
- Functions (What the system can do)
  - Use case diagram
- Time, dynamics, temporal constraints
  - Sequence (collaboration) diagram
  - Statechart diagram
  - Activity diagram

Class / object models

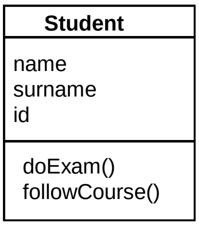
Object

- Model of entity (physical or inside software system)
  - ex.: student, exam, stack, window
- characterized by
  - identity
  - attributes (or data or properties)
  - operations it can perform (behaviour)
  - messages it can receive
- graphic representation: rectangle



Class

- Descriptor of objects with similar properties



SoftEng http://softeng.polito.it

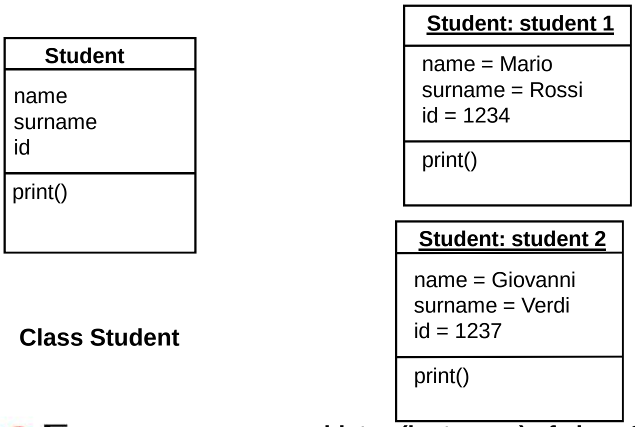
Class – cont.

- attribute
  - the name of an attribute is the same for all objects and can be described in the class
  - the value of an attribute may be different on each object and cannot described in the class
- operation
  - is the same for all objects and can be described in the class
  - will be applied to different object (possibly with different results)

SoftEng http://softeng.polito.it

Class and object

- object is instance of a class



Class Student

objetcs (instances) of class Student

SoftEng http://softeng.polito.it

Class and object: Java

```
class Student{
  String name;
  String surname;
  long int id;
  void print(){  System.out.println(“Info of
   student:” + “ ” + name + surname + id);
             }
}
```

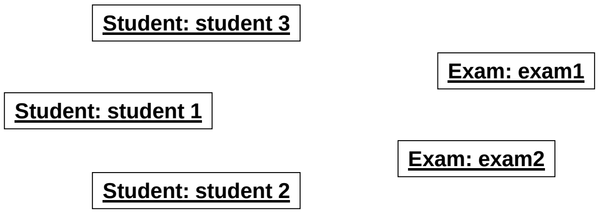
SoftEng http://softeng.polito.it

```
class Exam {
 int grade;
  Student s;
  void print(){
    System.out.println("Grade: " + grade);
  }
}
```

```
main(){
 Student student1;
 Student student2;
 student1 = new Student("Mario", "Rossi",
   1234);
 student2 = new Student("Giuseppe", "Verdi",
   1237);
 student1.print();
  student2.print();
}
```

Object diagram

- Models objects of interest in a specific case



  - Remark: above is a reduced notation for object/class
  - Remark: links are key part of diagram, see next slides

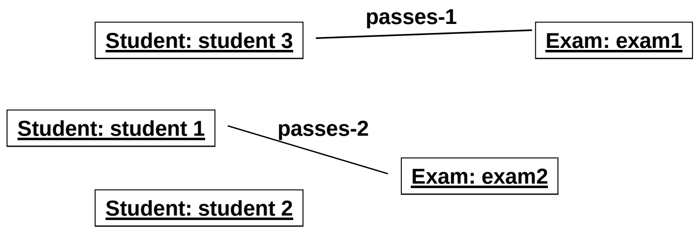
Class diagram

- Models classes of interest in a specific case

  - Remark: relationships are key part of this diagram, see next slides

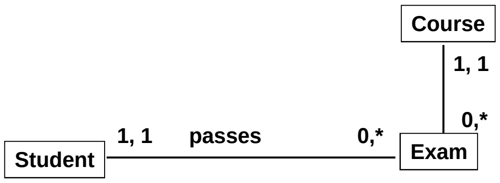
Link

- Model of association between objects

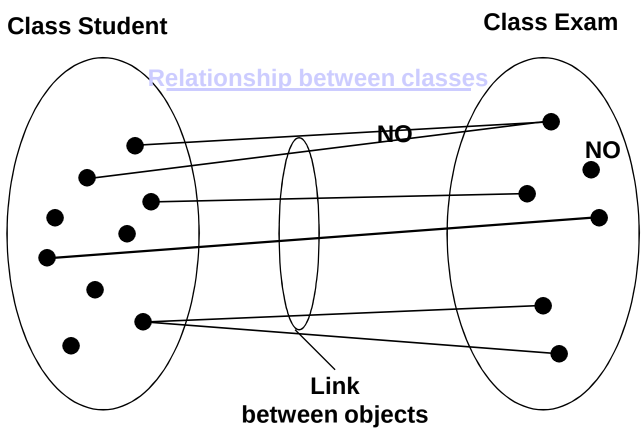


Relationship

- Descriptor of links with similar properties



Relationships



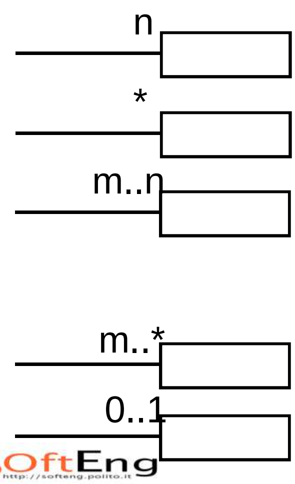
SoftEng

Multiplicity

- Constraint on max / min number of links that can exit from an object

SoftEng

Multiplicity



Exactly n

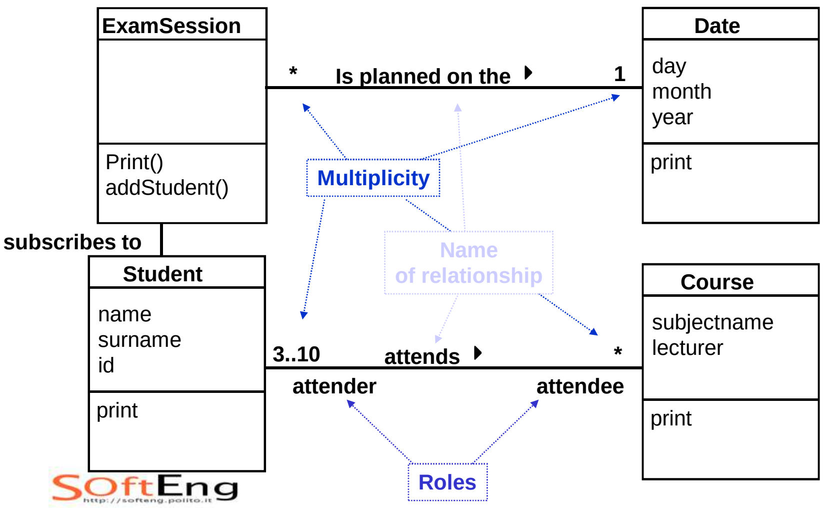
Zero or more

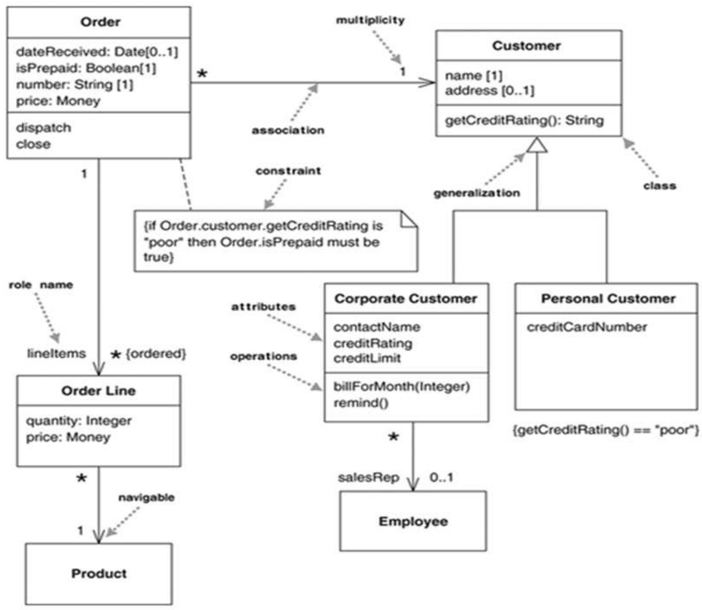
between m and n (m,n included)

from m up

Zero or one (optional)

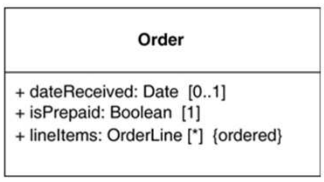
Relationships



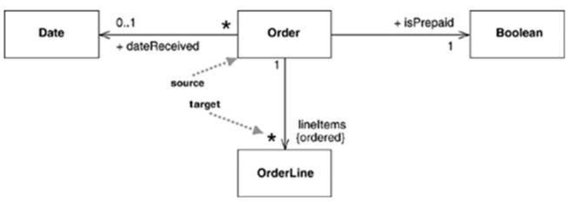


Properties

Showing properties of an order as attributes



Showing properties of an order as associations



SoftEng

Bidirectional Associations

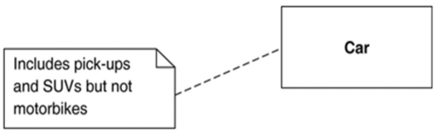
A bidirectional association

Using a verb phrase to name an association

SoftEng

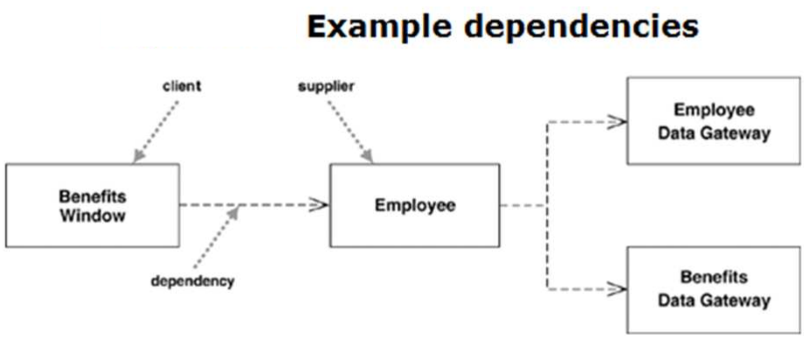
Notes and Comments

A note is used as a comment on one or more diagram elements



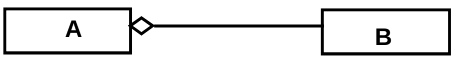
SoftEng

Dependency

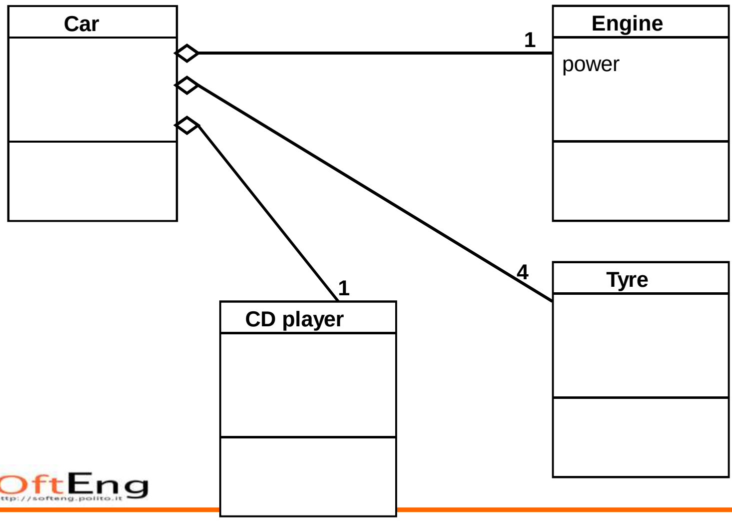


Aggregation

- B *is-part-of* A means that objects described by class B can be attributes of objects described by A



Example




```
Class Car {
  Tyre t[4];
  Engine e;
  CDPlayer cd;
}
class Tyre {}
Class Engine {}
```

Specialization

- or Generalization, or is-a

- A *specializes* B means that objects described by A have the same properties (attributes, operations) of objects described by B
- Objects described by A can have additional properties

SoftEng http://softeng.polito.it

Subclass superclass

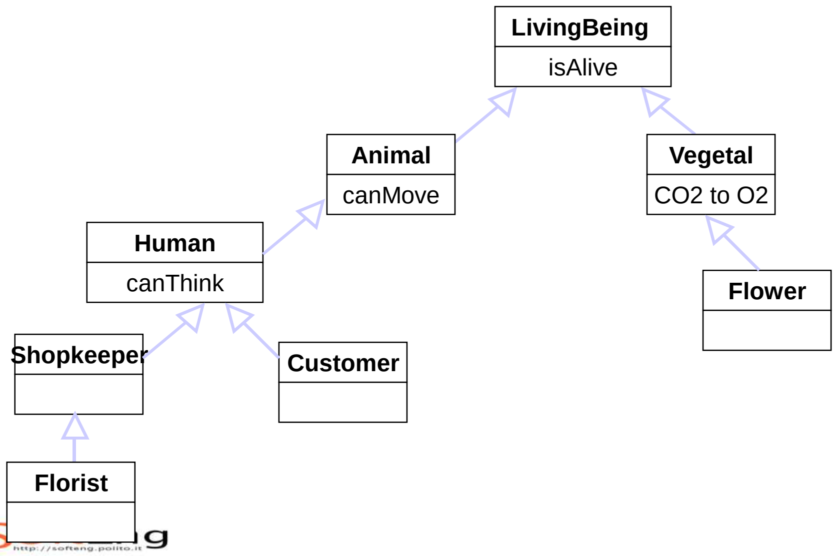
- Subclass = specialized class
- Superclass = generalization class

SoftEng http://softeng.polito.it

Inheritance

- mechanism associated to specialization/generalization relationship
- properties defined by B are inherited by A
  - A does not need to repeat these properties
  - Human
    - canThink (own property)
    - canMove (inherited from Animal)
    - IsAlive (inherited from LivingBeing)

Example



In short

- Object diagram (models)
  - object
  - link
- Class diagram (descriptors)
  - class
  - relationship
    - aggregation
    - specialization
  - multiplicity

- to model structural information
  - structural viewpoint

SoftEng http://softeng.polito.it

DO NOT in class diagrams

- Use plurals for classes
  - Classroom yes, no classroomS
- Use transient relationships
  - (they will be modeled in scenarios)
- Checkout loops
- multiplicities

SoftEng http://softeng.polito.it

DO NOT in class diagrams

- Repeat as an attribute of a class a relationship starting from the class
- Confound system design and glossary

SoftEng http://softeng.polito.it

Support from OO prog. languages

- object, class
  - supported
- relationships
  - aggregation
    - supported partially
  - specialization
    - supported

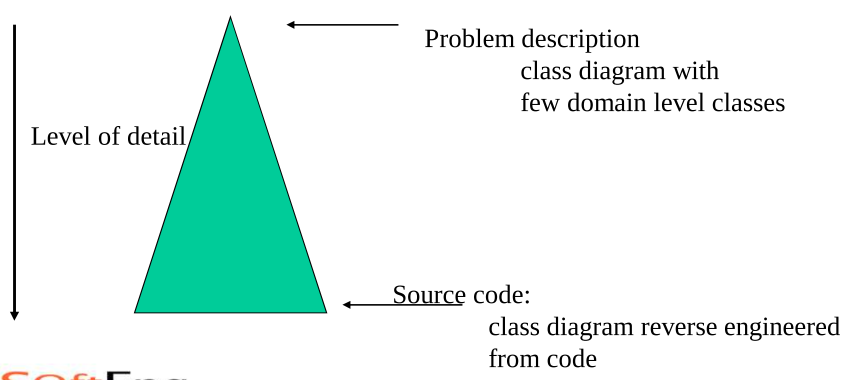
SoftEng http://softeng.polito.it

Use of class diagrams

- Class diagrams are just a notation
- can be used in different documents with different goals
  - user requirements
  - developer requirements
  - system design
  - (unit design)

SoftEng http://softeng.polito.it

Use of class diagrams



SoftEng http://softeng.polito.it

Use cases

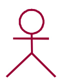
Use Cases

- Semi-formal notation
- Study of the application domain
- Identification of boundaries and interactions among the system and the external world
- Useful to
  - Oblige the analyst to state well-defined boundaries between system and external world
  - Organize system functions into elements (use cases) on which attention is focused
  - Supply a first basis for the specification of system structure from the user perspective

Use Case Diagram

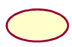
- Provide a more functional view of a software system
  - functions, actors
  - boundary
- Readable by customer/user
- Usually defined before class diagrams
- Diagram composed of actors, use cases, relationships

Elements of a Use Case



Actor

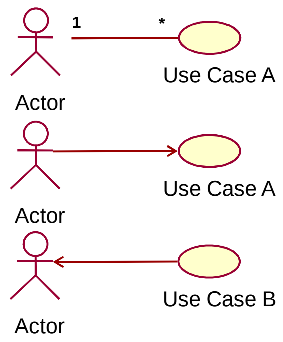
- Someone (user) or something (external system, hardware) that
  - Exchanges information with the system
  - Supplies input to the system, or receives output from the system

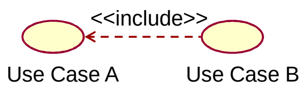


Use Case

- A functional unit (functionality) part of the system

Relationships

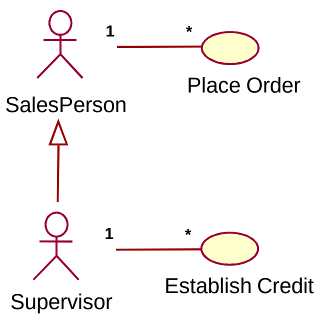




- Association models:
  - Which actors participate in a use case
  - Where execution starts
  - Adornments (e.g. multiplicity, direction) allowed
  - Actor1 participates in Use CaseA and is the trigger of the use case
  - Actor2 participates in UseCaseB and UseCaseB is the trigger
- Include
  - Models that functionality A is used in the context of functionality B (one is a phase of the other)

SoftEng http://softeng.polito.it

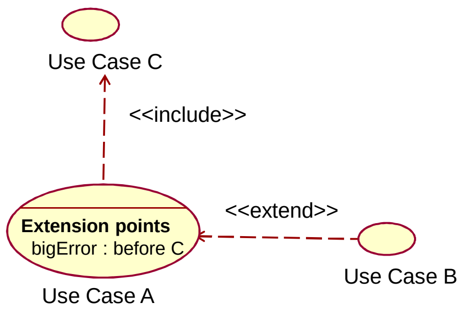
Relationships: generalization



- Generalization
  - Defines functionality B as a specialization of functionality A (e.g. a special case)
- Generalization
  - A generalization from an actor B to an actor A indicates that an instance of B can communicate with the same kinds of use-case instances as an instance of A

SoftEng http://softeng.polito.it

Relationships: extension

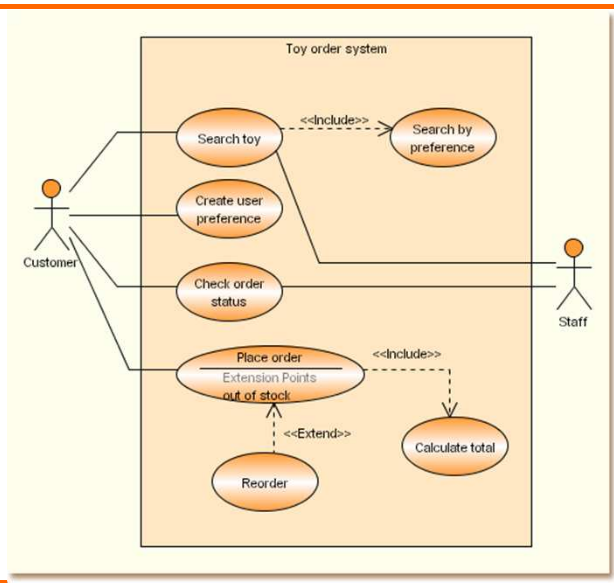


- Extension
  - An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented by the behavior specified by A
  - The behavior is inserted at the location defined by the extension point (name : where) in B, which is referenced by the extend relationship

SoftEng

73

Use case – Example



SoftEng

74

Use case

- A scenario is a sequence of steps describing an interaction between a user and a system
- A use case is a set of scenarios tied together by a common user goal.

SoftEng http://softeng.polito.it

Use cases vs.requirements

- Requirement (functional)
- Use case
  or
  scenario in use case
  or
  step in scenario

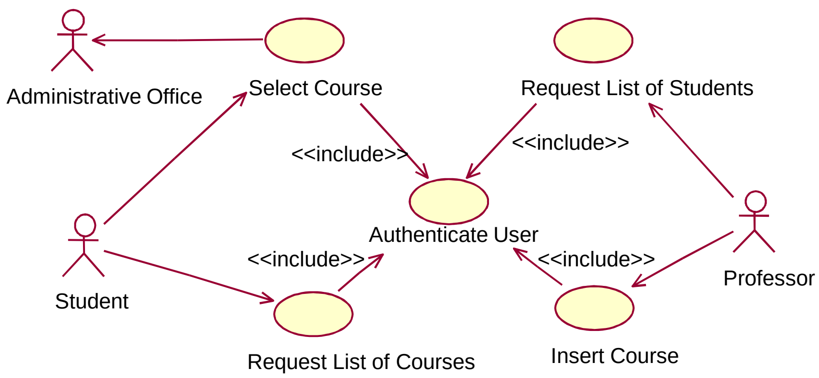
- Mapping is not 1:1
- Requirement purpose is to support traceability and tends to be finer grained than use case
- Use case purpose is to understand how system works

SoftEng http://softeng.polito.it

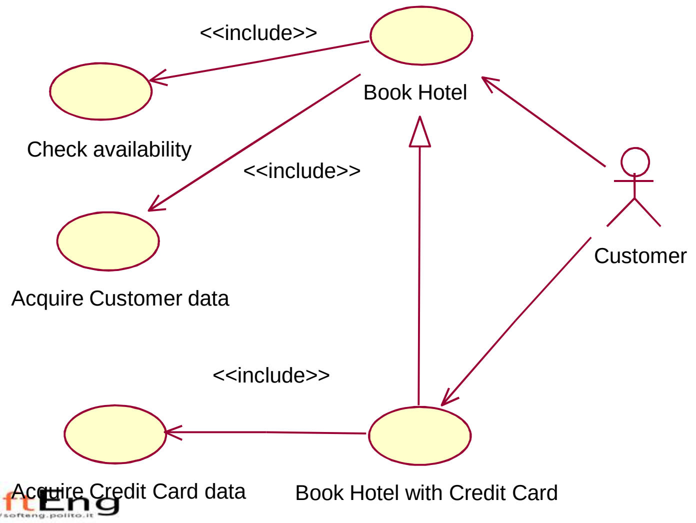
Example: student management

- students select courses
- professors update the list of available courses
- professors plan exams for each course
- professors can access the list of students enrolled in a course
- professors perform exams then record issue of exam for student (pass/no pass, grade)
- all users should be authenticated

Example



Example



Use case diag and class diagram

- They must be consistent
  - Use case diagram
    - actor
    - use case
    - interaction
  - Class diagram
    - may become a class
    - must become one operation on a class – may originate several operations on several classes (see sequence diag)
    - not represented (see dynamic diagrams)

Dynamic models

- Sequence diagrams
- Collaboration diagrams
- State charts
- Activity Diagrams

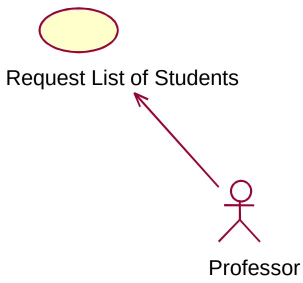
Sequence diagrams

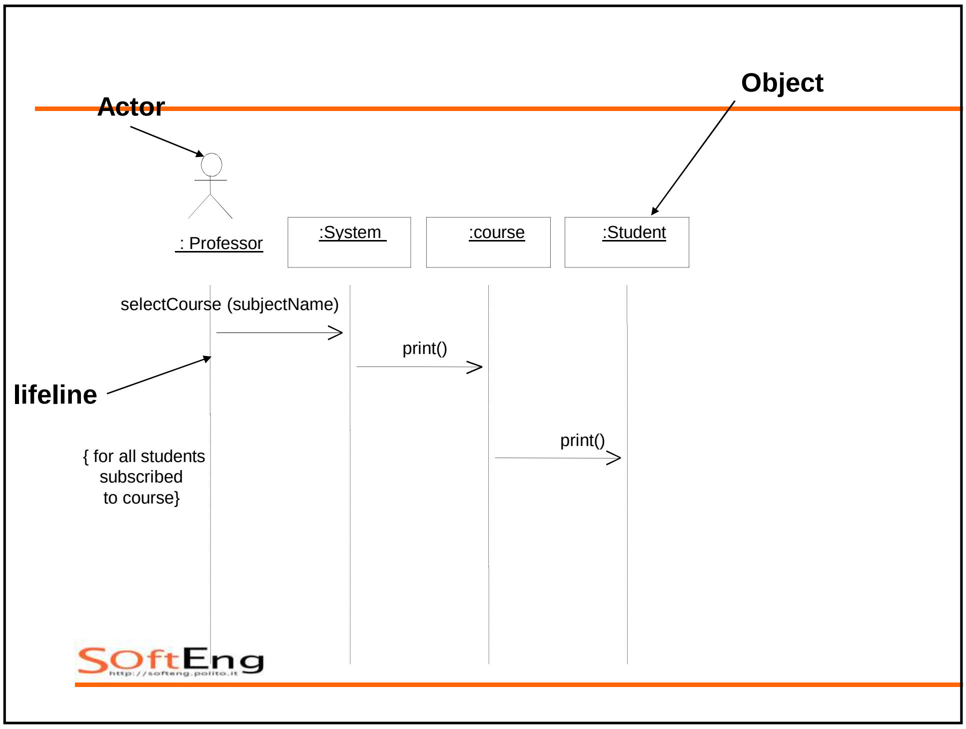
Sequence Diagrams

- One vertical line per object or actor
- Time passes top down
- Arrows represent message passing among objects

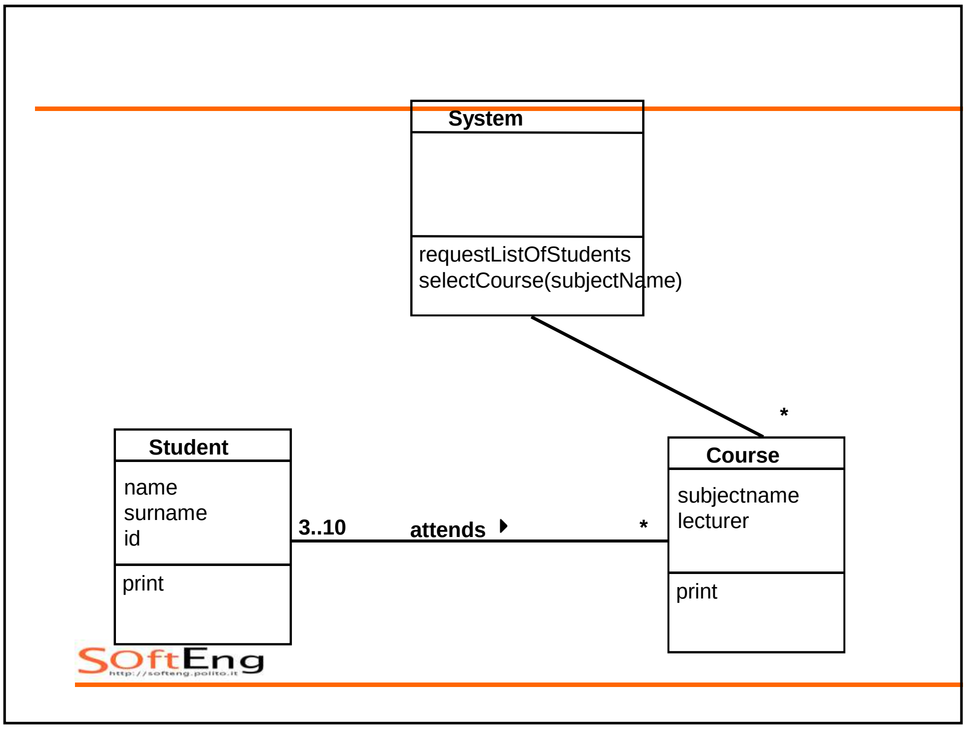
Ex. Starting from

- Use case “request list of Students”





Object
Actor
: Professor
:System
:course
:Student
selectCourse (subjectName)
print()
lifeline
print()
{ for all students subscribed to course}
SoftEng



System
requestListOfStudents
selectCourse(subjectName)
*
Student
name
surname
id
print
3..10
attends
*
Course
subjectname
lecturer
print
SoftEng

Sequence diag and Use case

- sequence diag corresponds to a Use case
  - provides detail on how Use case is executed
- Use case can be described by several sequence diagrams

Sequence diag and class diag

- all objects/classes appearing in sequence diagram must be defined in object/class diagram
- all messages sent to object/class must be defined as operation in receiving object/class

Use of sequence diagrams

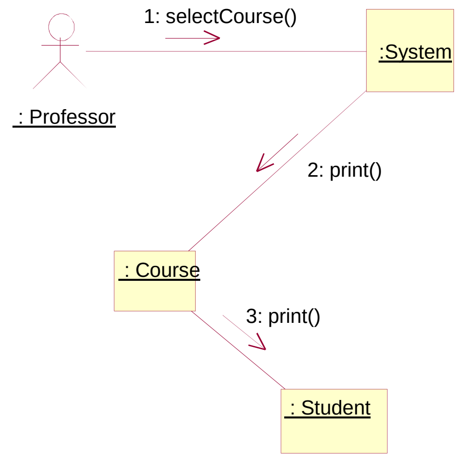
- One software system <--> several (infinite) sequence diagrams
- only the key ones can be described
  - starting from use cases
  - key functions, difficult functions, nominal cases, key exceptions

SoftEng http://softeng.polito.it

Collaboration diagrams

- Same (actually less in some cases) information and constraints as sequence diagrams

SoftEng http://softeng.polito.it



SoftEng

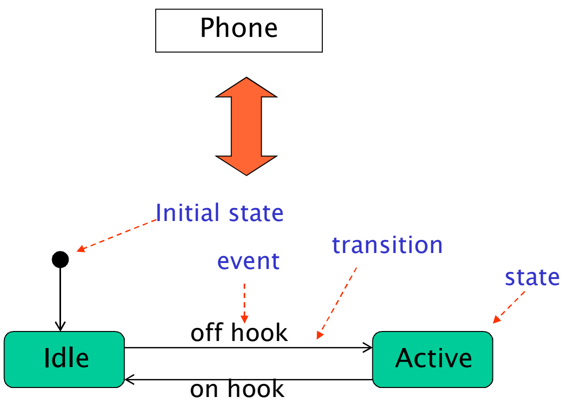
Statechart diagram

SoftEng

UML Statechart Diagram

- Shows the sequences of states that objects of a class go through during their life cycle in response to external events and also the responses and actions in reaction to an event.
- Model elements
  - States
  - Transitions
  - Events
  - Actions and activities

Example: STD for a Phone



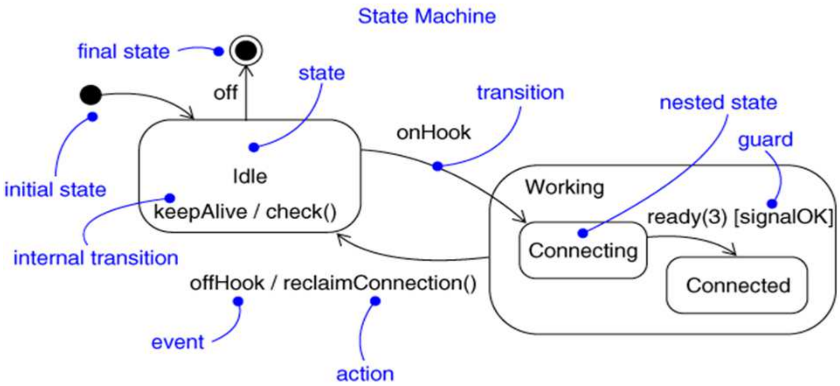
State Diagram

- Graph made of nodes and arcs
  - Nodes represent states;
  - arcs represent transitions between states
  - Arcs are associated to events, that trigger the transition
- Describes the behaviour of a single class of objects
- Can represent
  - one-shot life cycles (initial and final state)
  - continuous loops (no final state)

Classes that Need State Diagrams

- Not all classes need a state diagram
- State-dependent classes
  - objects described by the class react differently to events depending on their state
- **State-independent classes** do not need State Diagrams
  - an object always responds the same way to an event

Statecharts: glossary



SoftEng

Elements

- Actions – no time passes
  - Sending a message, change an attribute value, generate an output
- Activities – time passes
  - Doing a calculation, executing an algorithm, counting a time interval
- Events
  - Receiving a message, terminating a time interval
- States
  - Idle, busy, ..
- Transitions
  - Moving from a state to another state

SoftEng

State

- Abstraction of attribute values and links of an object
- Sets of values are grouped together into a state
- Corresponds to the interval between two events received by the object
  - events represent points in time
  - states represent intervals of time
- Has duration

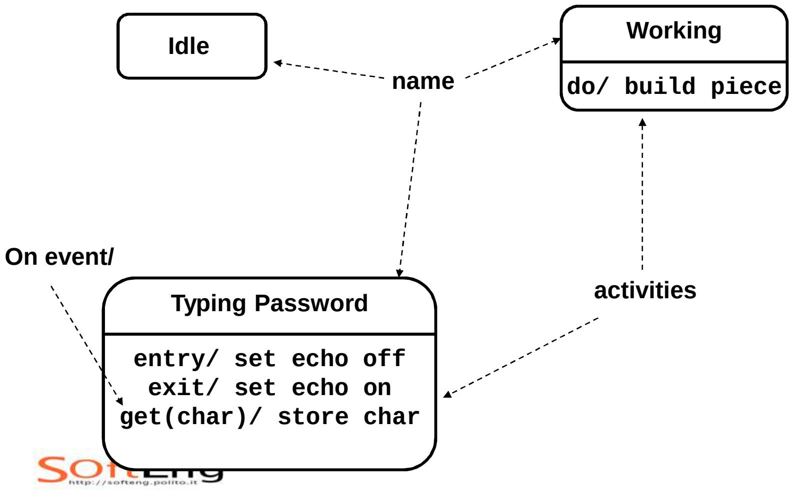
SoftEng
http://softeng.polito.it

State

- Characterized by
  - Name
  - Activities (executed inside the state)
    - **Do**/ activity
  - Actions (executed at state entry or exit)
    - **Entry**/ action
    - **Exit**/ action
  - Actions executed due to an event
    - Event [Condition] / Action ^Send Event

SoftEng
http://softeng.polito.it

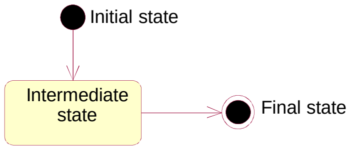
Notation for States



SoftEng http://softeng.polito.it

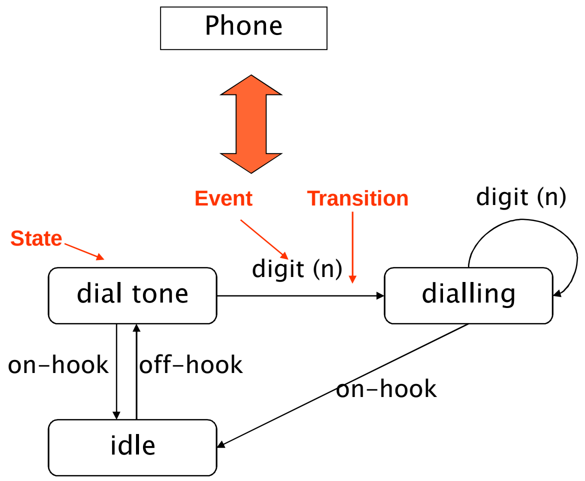
Notation for States (cont.)

- Termination states have special symbols
  - The initial state is unique, and models the state in which the object is initially
  - The final state(s) is a state in which the object terminates to execute



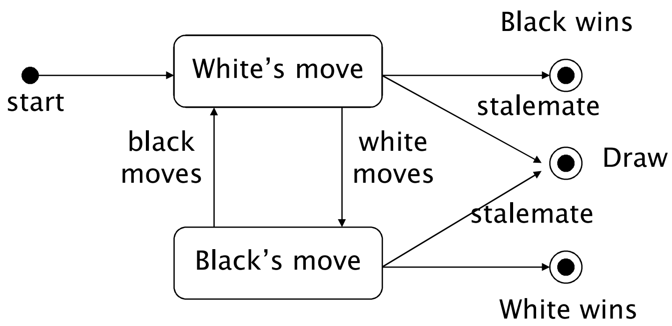
SoftEng http://softeng.polito.it

Example



SoftEng

Example

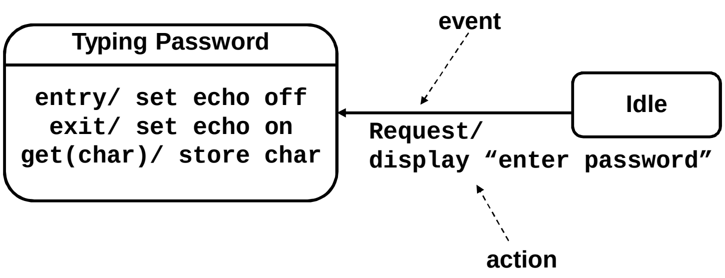


SoftEng

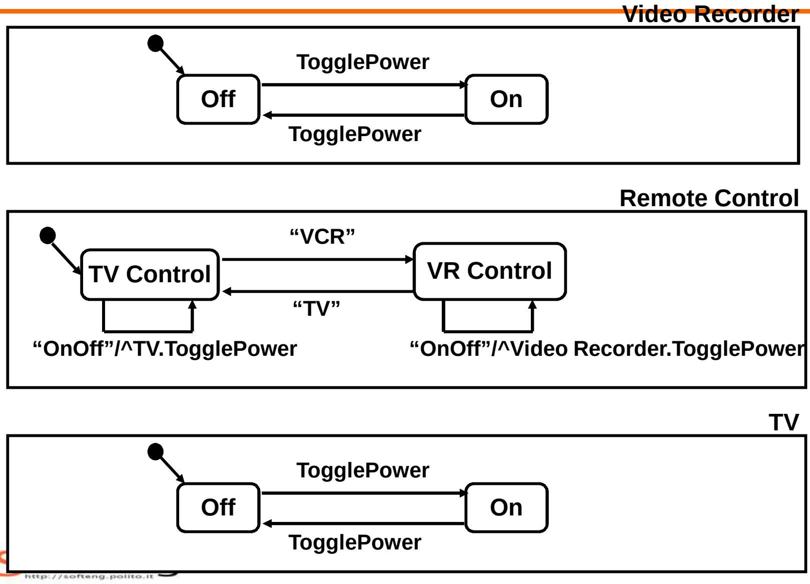
Transition

- Models a state modification
  - Occurs at the verification of an event, if a condition is valid
  - Can be associated with an action and/or a method of an object
- Is described according to the following syntax
  - Event [Condition] / Action ^Send Event

Transition



StateChart Example



Event Types

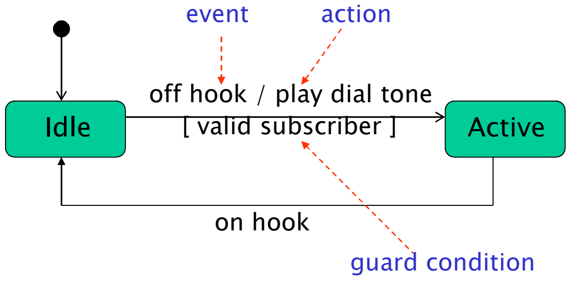
- External Event (also known as system event)
  - is caused by something outside the system boundary
  - e.g. when a cashier presses the "enter item" button on a POS, an external event has occurred.
- Internal Event
  - is caused by something inside our system boundary.
  - In terms of SW, an internal event arises when an operation is invoked via a message sent from another internal object. (The messages in collaboration diagrams suggest internal events)
- Temporal Event
  - is caused by the occurrence of a specific date and time or passage of time.

SoftEng http://softeng.polito.it

Guard Condition

- Boolean function of object values
- Valid over an interval of time
- Can be used as guards on transitions
- Guard condition shown in brackets, following event name

Transition Action and Guards



Operations

- Attached to states or transitions
- Performed in response to corresponding states or events
- Types
  - Activity
  - Action

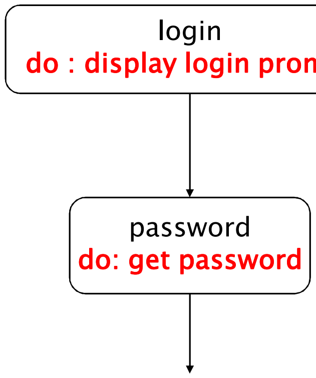
SoftEng

Operations: Activity

- Activity
  - operation that takes time to complete
  - associated with a state
  - include continuous or sequential operations
  - notation “do: A” within a state box
    - indicates activity A
    - starts on entry
    - ends on exit

SoftEng

Example – State Activities



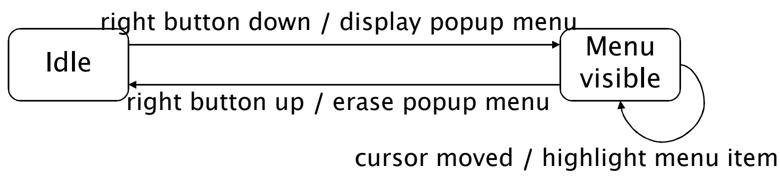
SoftEng http://softeng.polito.it

Operations: Action

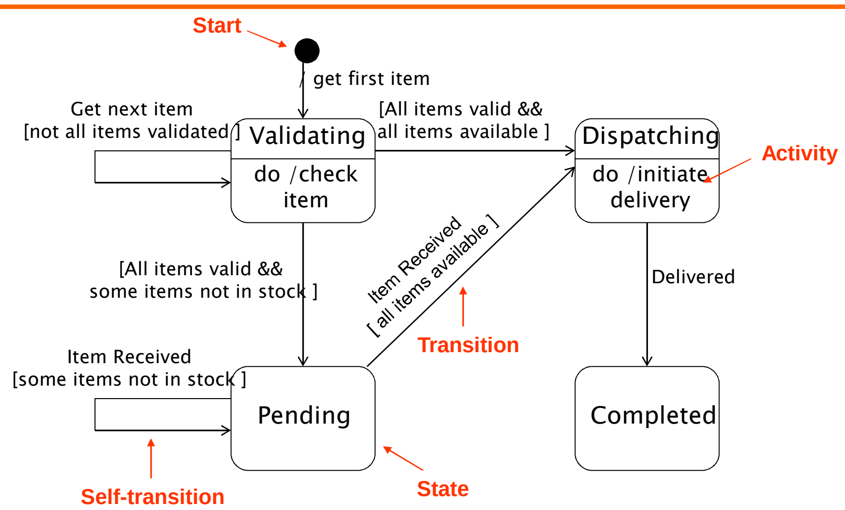
- **Action**
  - instantaneous operation
  - associated with an event
  - notation
    - slash (“/”) and name of the action, following the event

SoftEng http://softeng.polito.it

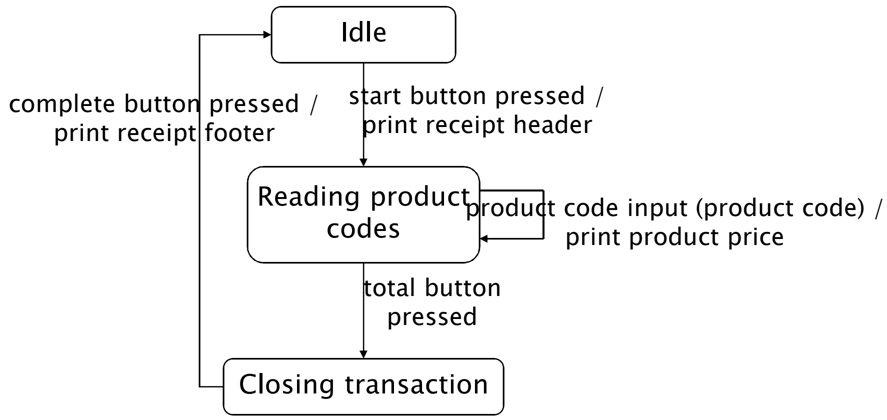
Example – Transition Actions



Statechart Example



Example



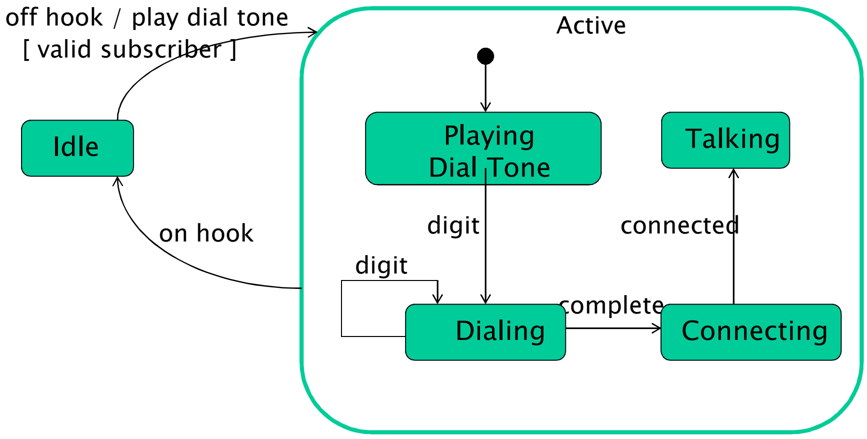
SoftEng

Nested State Diagrams

- State diagrams can get complex
- For better understanding and management
- A State in a state diagram can be expanded into a state diagram at another level
- Inheritance of transitions

SoftEng

Example: Nested States



Activity diagram

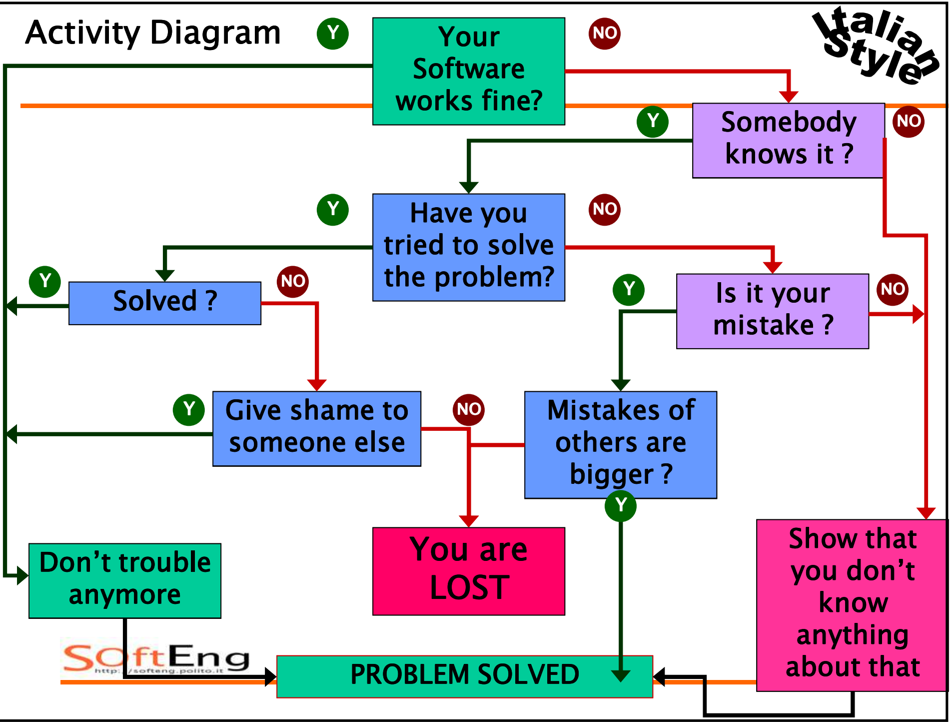
Activity Diagram

- Extension of Statechart Diagram used to represent temporal sequence of activities and data flow
- Used to represent workflow process, or the inner service logic of an algorithm or function, process
- Parallel process representation and synchronization (fork – join)
- Partial Fork and Join are not definable

SoftEng http://softeng.polito.it

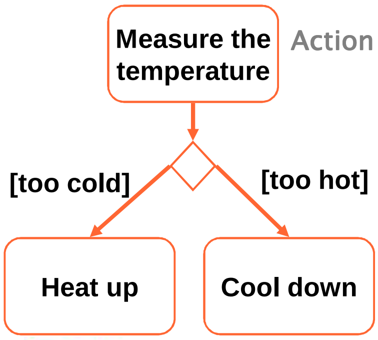
121

Activity Diagram

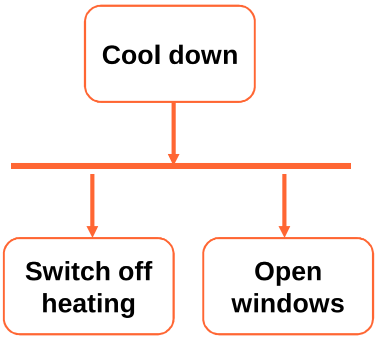


Action state e Transition

Decisione:
Alternativa in base a condizione

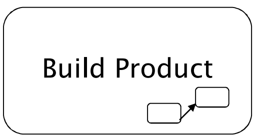


Sincronizzazione tra flussi di controllo paralleli

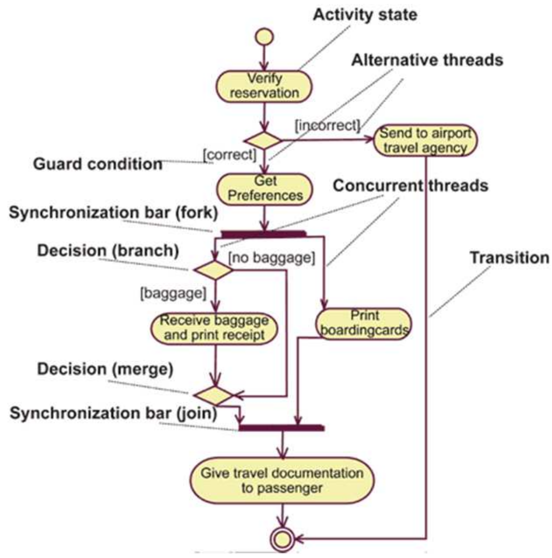


SoftEng http://softeng.polito.it

Activity diagrams: glossary

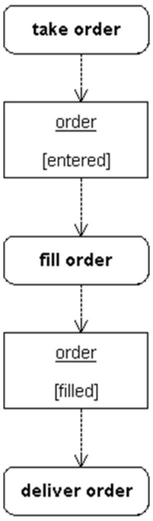


Subactivity state

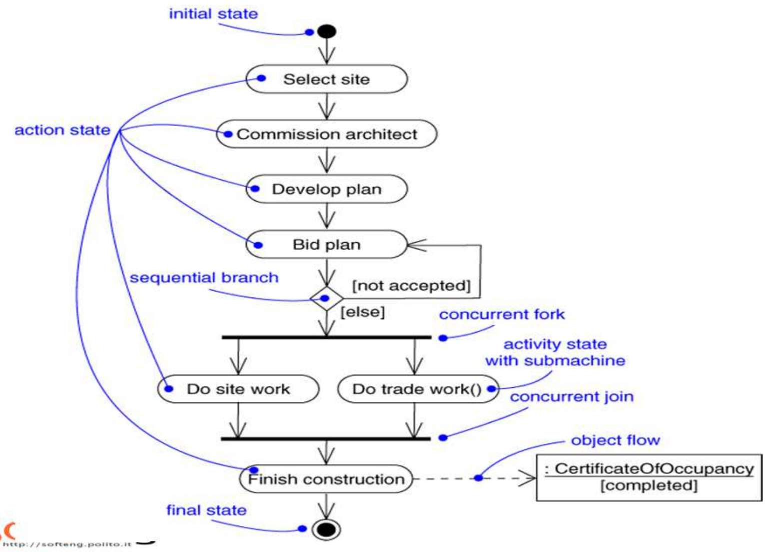


SoftEng http://softeng.polito.it

Object Flow

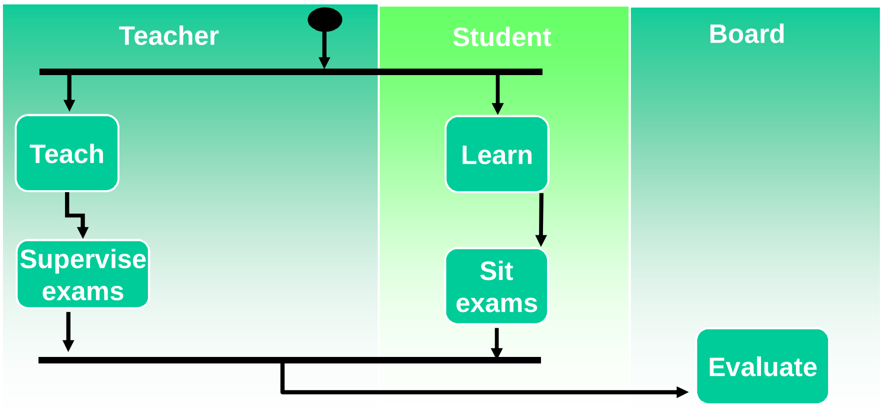


Example I



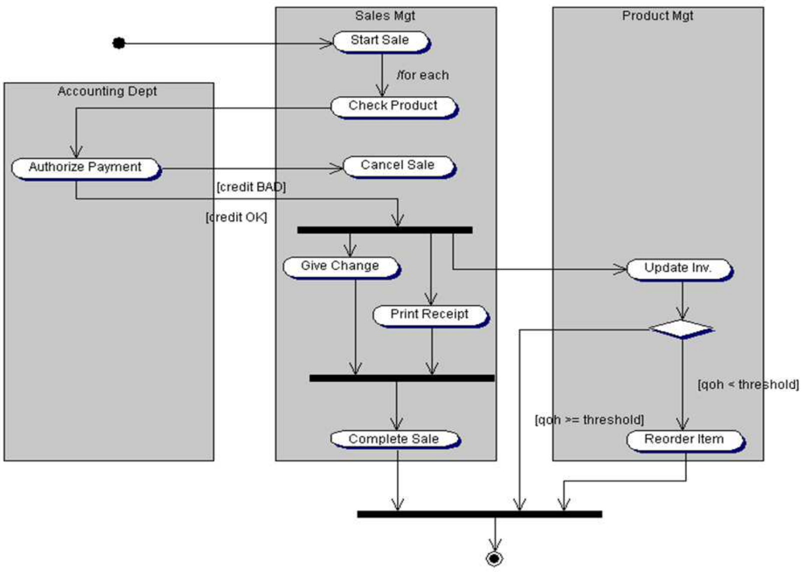
Swim-lanes

Linee verticali opzionali in un activity diagram che mostrano l'allocazione di un gruppo di attivita` a un ruolo o persona dell'organizzazione



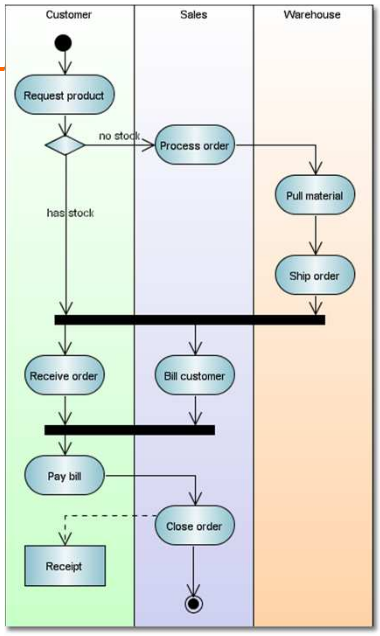
127

Example II



128

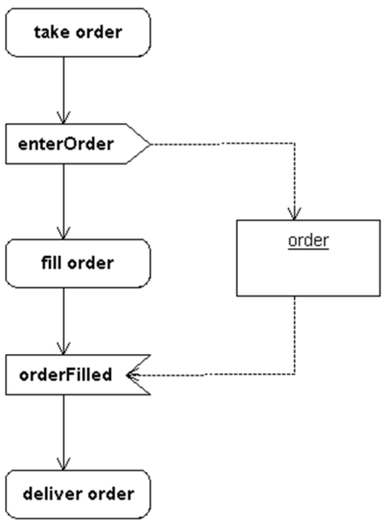
Example III



SoftEng

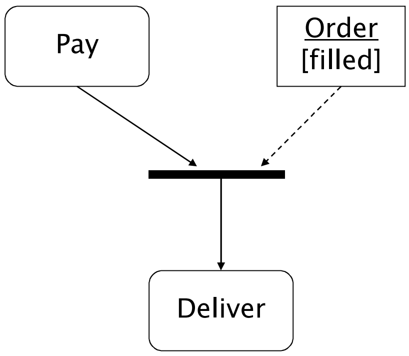
Signals

- Signal sending between transition
  - Optionally to an object
- Signal receiving
  - Transition governed by signal



SoftEng

More on sync



SoftEng http://softeng.polito.it

Physical Diagrams

SoftEng http://softeng.polito.it

UML Physical Diagrams

- Component diagram
  - Various components in a system, and their dependencies
  - Explains the structure of a system
- Deployment diagram
  - Physical relationships among software and hardware in a delivered systems
  - Explains how a system interacts with the external environment
- Package Diagram
  - High Level System Architecture

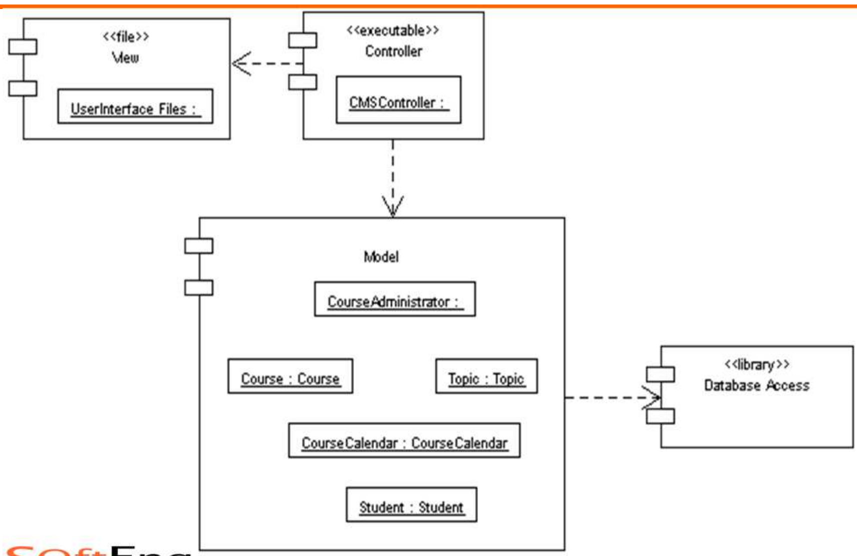
SoftEng http://softeng.polito.it

Components

- They are used to model different kind of components in a system:
  - packages
  - Executable files
  - System or Application Libraries
  - Files
  - Tables in a DB
  - …

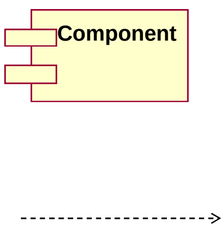
SoftEng http://softeng.polito.it

Example



SoftEng

Component Diagram

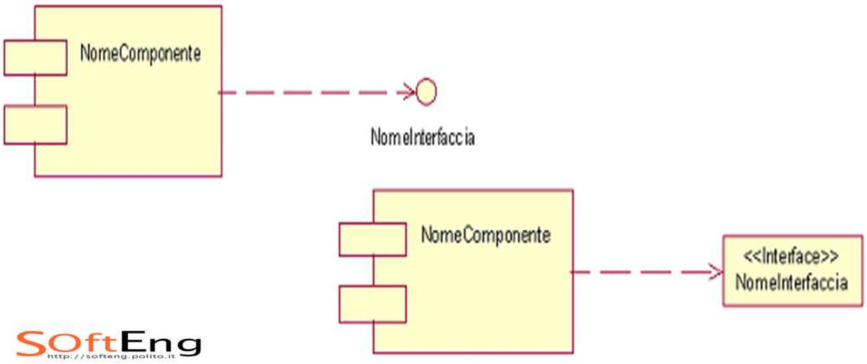


- Physical module of code
  - A package
  - A collection of classes
- Dependency among components
  - Change impact
  - Communication dependency
  - Compilation dependency

SoftEng

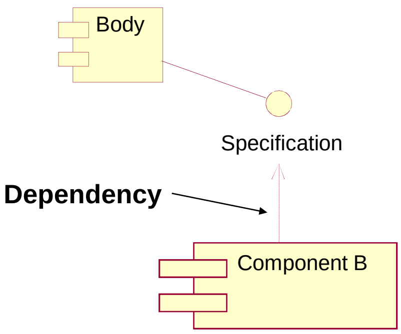
Elements of component diagram

- A Component Interface can be represented in two ways



SOftEng
http://softeng.polito.it

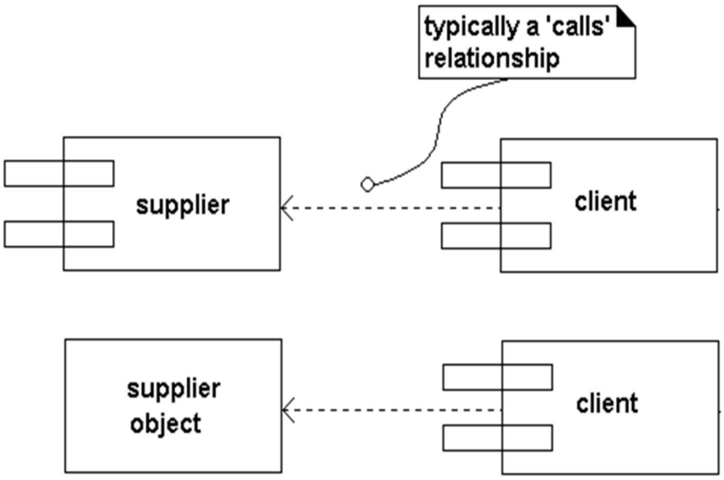
Component Diagram



- Other elements
  - Processes
    - Contained into components
    - Thread, process,...
  - Programs
    - Language-dependent
    - Applet, Application,...
  - Subsystems
    - Organising components into (nested) packages

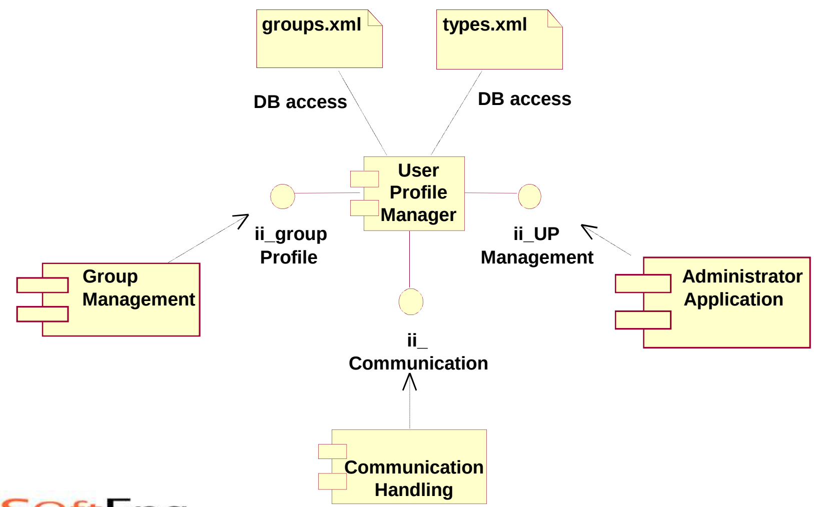
SOftEng
http://softeng.polito.it

Dependency: run-time Relationship

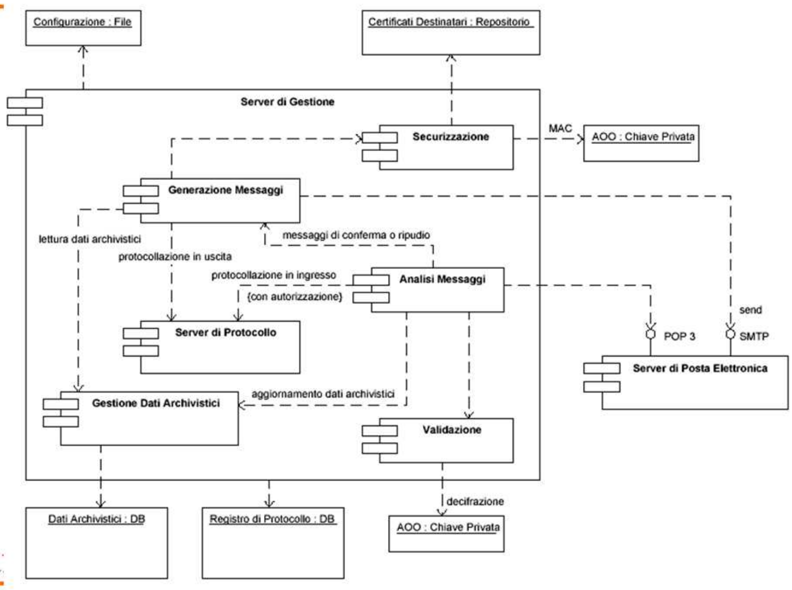


SoftEng

Example

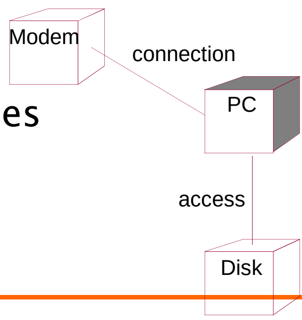


SoftEng

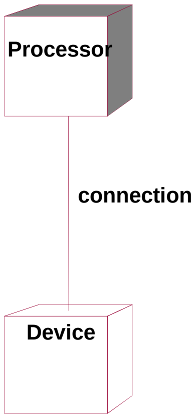


Deployment diagram

- Physical layout of the various hardware components (nodes)
  - Processor: capable of executing programs
  - Device: component with no computing power
- Distribution of
  - Executable
  - Programs on nodes
- Node classes and instances

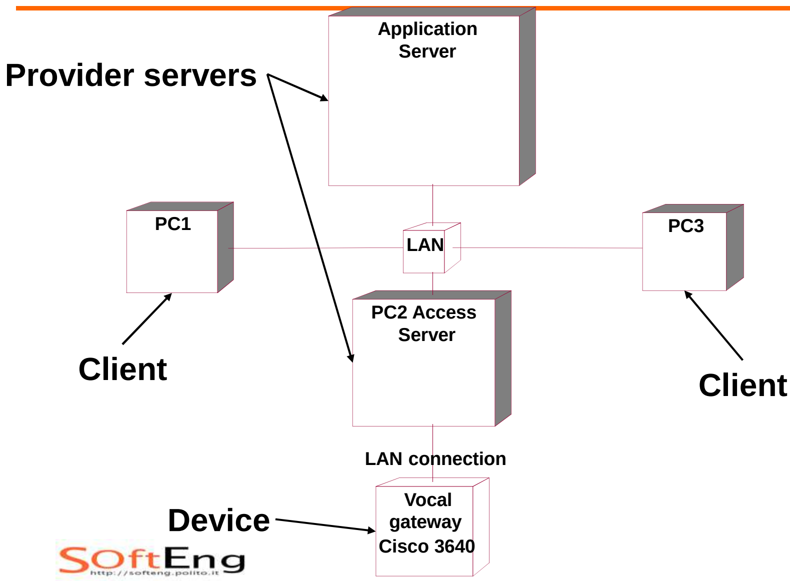


Deployment diagram



- Nodes
  - Some kind of computational unit
  - Programmable resource
    - Where components can be executing
  - Hardware resource
    - Usable by components
- Connections
  - Communication paths over which the system will interact

Example

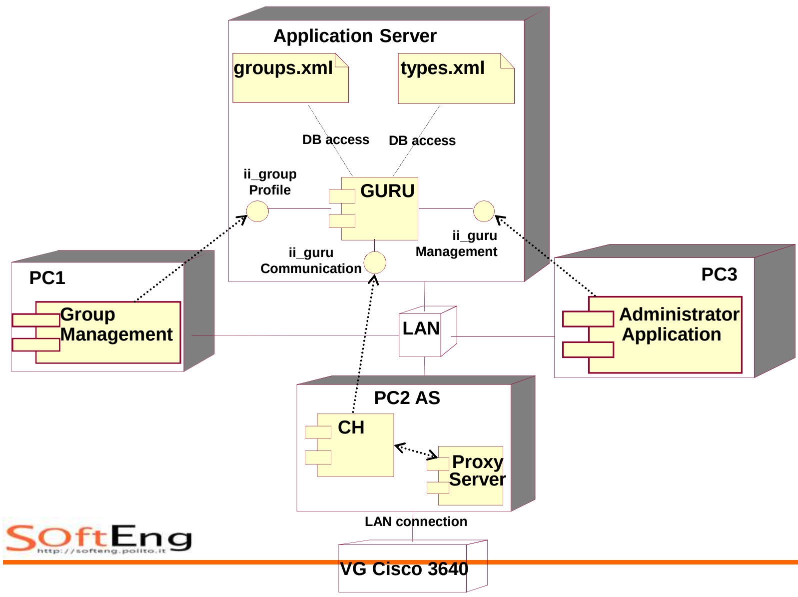


Combining the two Diagrams

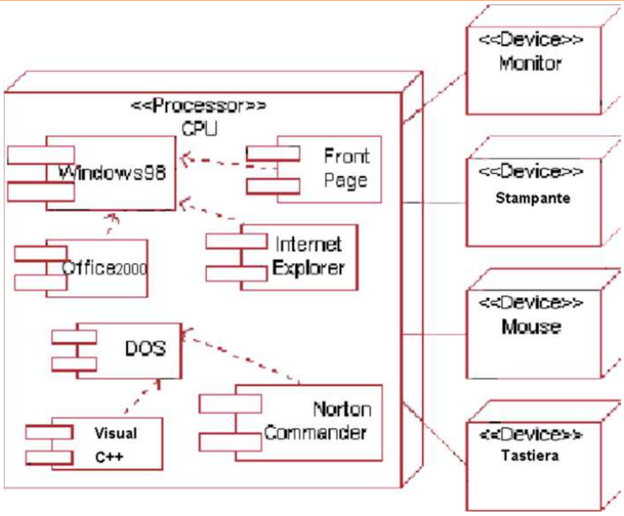
- Place the Component Diagram on top of the Deployment Diagram
  - Which components run on which nodes?
    - System awareness of components
    - Component↔Interface communication details

SoftEng http://softeng.polito.it

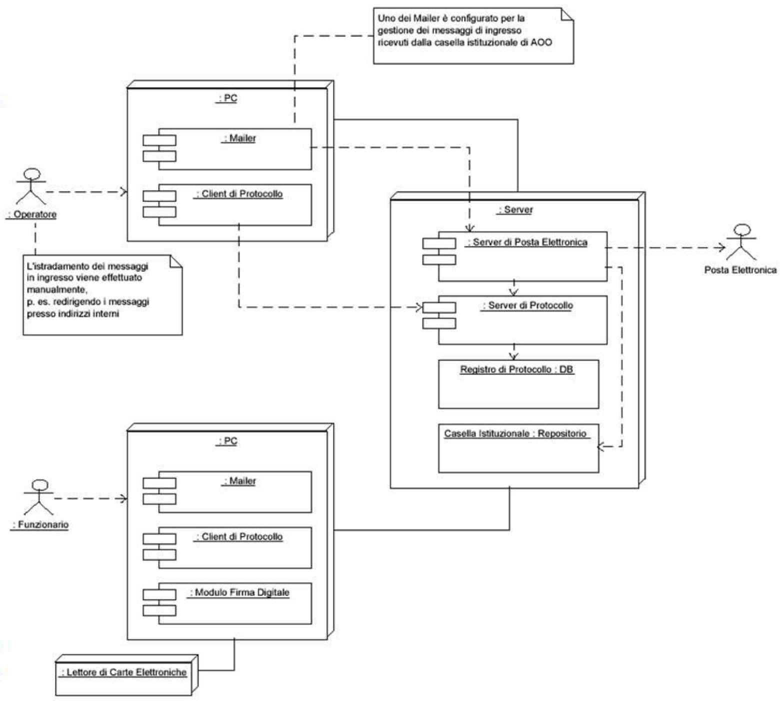
Example



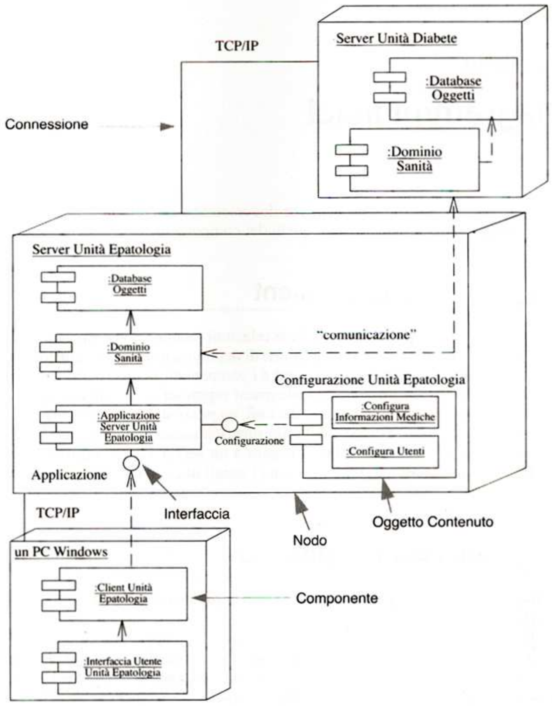
SoftEng http://softeng.polito.it



<<Processor>>
CPU
Windows98
Front Page
Office2000
Internet Explorer
DOS
Norton Commander
Visual C++
<<Device>>
Monitor
<<Device>>
Stampante
<<Device>>
Mouse
<<Device>>
Tastiera



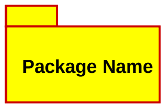
Uno dei Mailer è configurato per la gestione dei messaggi di ingresso ricevuti dalla casella istituzionale di AOO
: PC
: Mailer
: Client di Protocollo
: Operatore
L'istradamento dei messaggi in ingresso viene effettuato manualmente, p. es. redirigendo i messaggi presso indirizzi interni
: Server
: Server di Posta Elettronica
: Server di Protocollo
Registro di Protocollo : DB
Casella Istituzionale : Repositorio
Posta Elettronica
: PC
: Mailer
: Client di Protocollo
: Modulo Firma Digitale
: Funzionario
: Lettore di Carte Elettroniche



UML Package

- Package are containers of other UML elements
- Package defines a scope for its elements
- Package can be nested
  - tree-structure, like a file-system

UML Package Diagrams

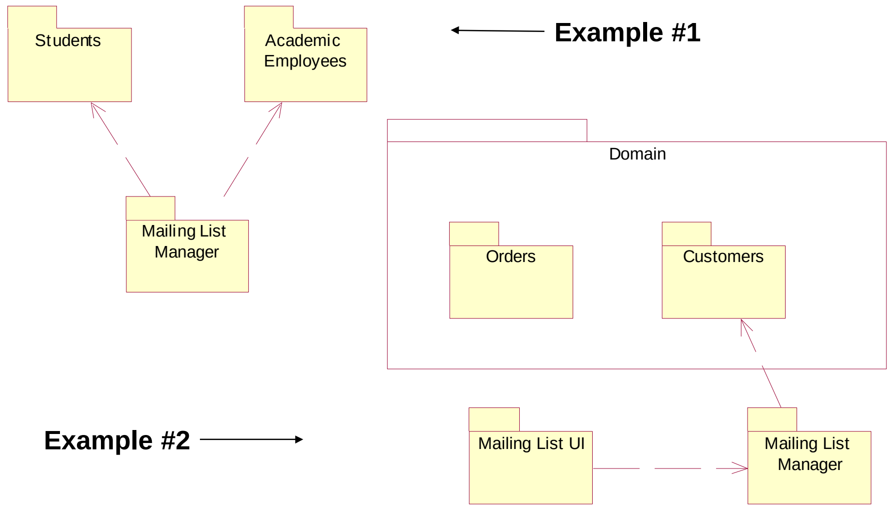


- Notation:
  - A Package box with a name
  - Dependency (it groups relationships among classes in different packages)

Package Diagram

- It is any diagram showing packages of classes and the dependencies among them
- It is just a Class Diagram showing only packages and inter-dependencies
- Dependency
  - Changes to the definition of one element may cause changes to the other
  - Message sending, structural composition, usage,…

Examples of Package Diagram



SoftEng
http://softeng.polito.it

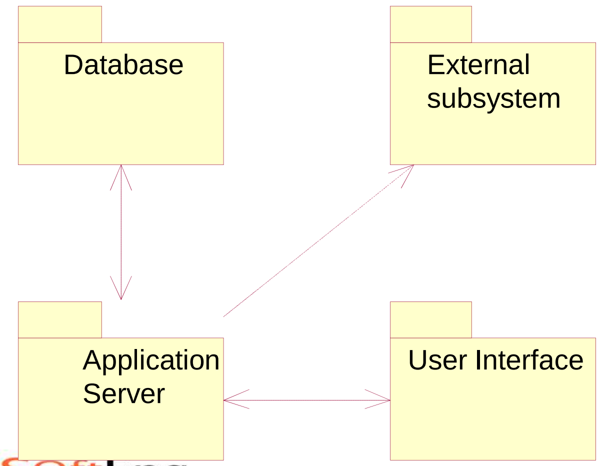
How to use packages

UML packages can be used in 2 ways:

1. During analysis to draw a high level architecture
   - Grouping classes in subsystems
   - Underline subsystems dependencies
2. During design in package diagram to organize a complex class diagram

SoftEng
http://softeng.polito.it

Package Diagram as Software Architecture



SoftEng http://softeng.polito.it

Package Diagram

- It is a class diagram where packages are used to group sets of classes
- Use it when there are lots of classes
- The package interface is the set of all interfaces of contained classes

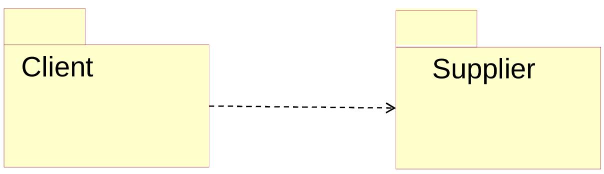
SoftEng http://softeng.polito.it

Identify a Package

- Given a big class diagram it is often needed to introduce packages to clarify the diagram
- Group together classes offering similar functionalities, with a high coupling among classes within the same package.
- A good package organization may lead to a low coupling between packages

Package vs classes

- Only these relationships are allowed:
  - Dependency
  - Realization
- Read as: client *depends on* supplier



Package dependencies

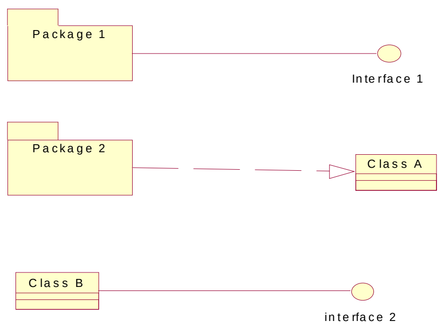
- Dependency between packages means inner classes in “*client*” package can :
  - Inherit from
  - Instantiate
  - Use (invoke methods of)

  …classes in “*supplier*” package
- Goal: Minimize the number of dependencies among packages

Realization relationship

- It is a relationship in which client realizes (implements) operations defined by supplier
- These are valid notations:

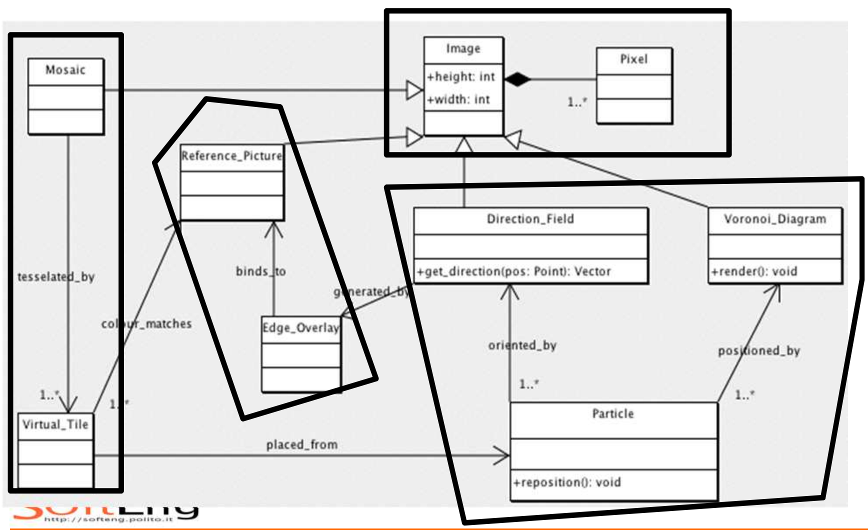
From:	To:
– Package	Interface
– Package	Class
– Class	Interface



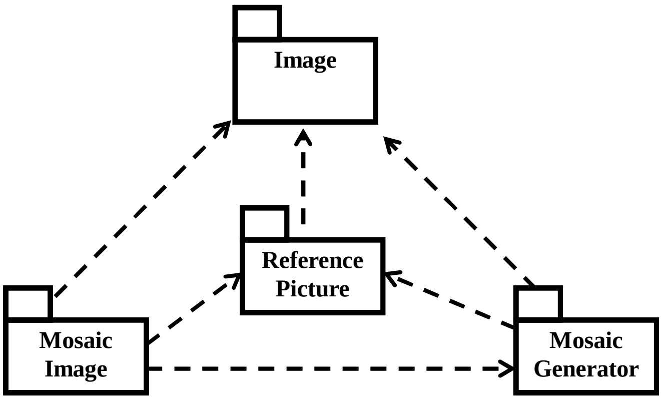
From Class to Package diagram

1. Aggregate classes related to same functionalities in the same package.
2. Classes in the same inheritance hierarchy typically go in the same package.
3. Classes related by aggregation or composition relationships typically go in the same package
4. Classes collaborating may go in the same package
   - Have a look to Sequence diagrams

Example: Class ➔ package



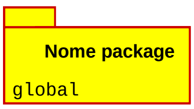
Example: Package Diagram



SoftEng http://softeng.polito.it

Package Global

- The label “global” means that a package can be used by all other packages in the system.
- E.g. the package contains many utility classes used by all other packages.
- Dependencies with global package are no more depicted
  - The package diagram is more readable



SoftEng http://softeng.polito.it

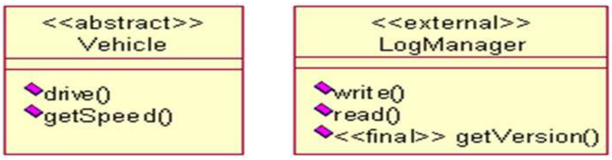
UML Profile

- UML defines how to extend the standard adding a new semantics to model elements
- UML Profiles are used to meet specific modeling requirements for
  - A specific domain (ex: business modeling, telecom, security,...)
  - A specific technology (ex: UML-EJB, Web)
- A UML Profile uses 3 UML extension mechanisms:
  - Stereotypes
  - Properties (Tagged Values)
  - Constraints (with Object Constraint Language)

SoftEng http://softeng.polito.it

Stereotype

- Give a different semantics to a model element (typically a class element)
  - Standard Stereotypes:
    - Interface, Abstract, Subsystem
  - Stereotype in UML Profiles
    - EJB in UML-EJB profile
    - Busineetc...



SoftEng http://softeng.polito.it

Stereotype Package: Subsystem

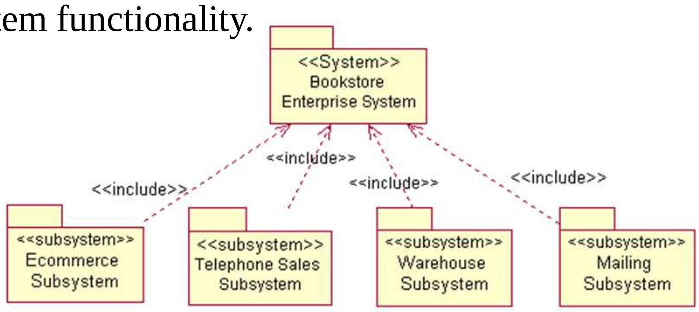
- A subsystem should be used when a set of classes and/or other packages need to be encapsulated within a container and hidden behind a set of well-defined interfaces.
- None of the contents of subsystem are visible except the interfaces of the subsystem.
- This allows subsystems to be easily replaced, and the implementations changed, provided the interfaces remain unchanged.
- It offers a degree of encapsulation greater than that of the package.

SoftEng

Example

Bookstore system is composed by 4 subsystems with different functionalities.

Noted throughout the <<include>> relationships, each subsystem provides a certain piece of the Bookstore system functionality.



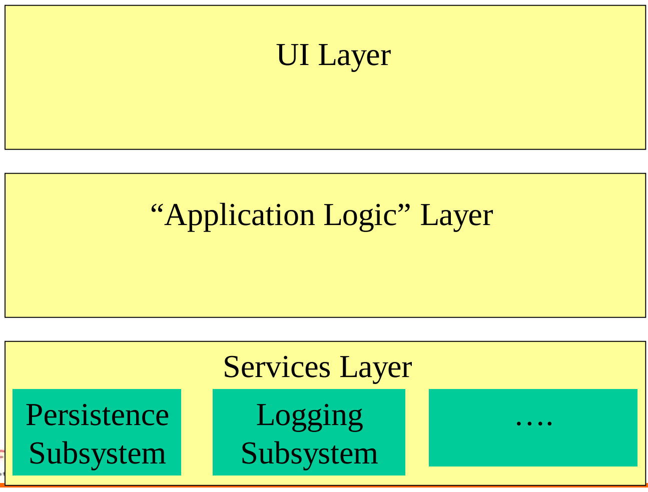
SoftEng

Architecture Design

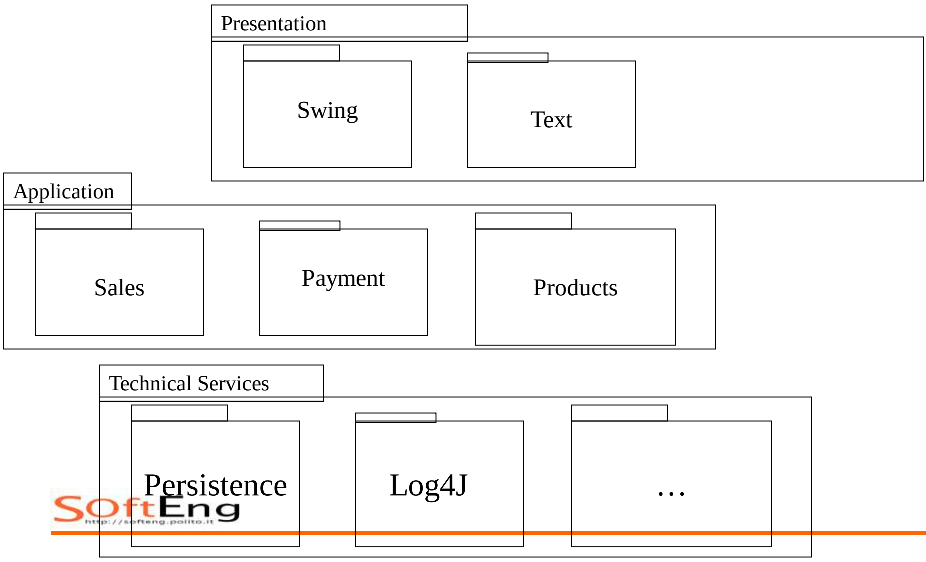
- System design often follows top-down approach: the abstract view is further refined in subsystems
- A software architecture can be divided in:
  - Layers
  - Subsystems
  - Packages (or Software Modules)
- Packages (or the subsytem stereotype) can be used to draw the system architecture

SoftEng

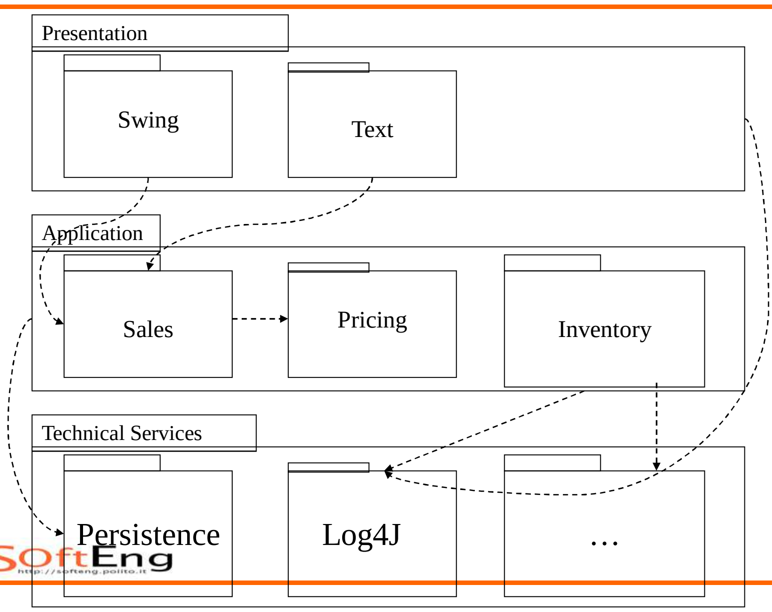
Example or architectural layers



UML Package Diagram



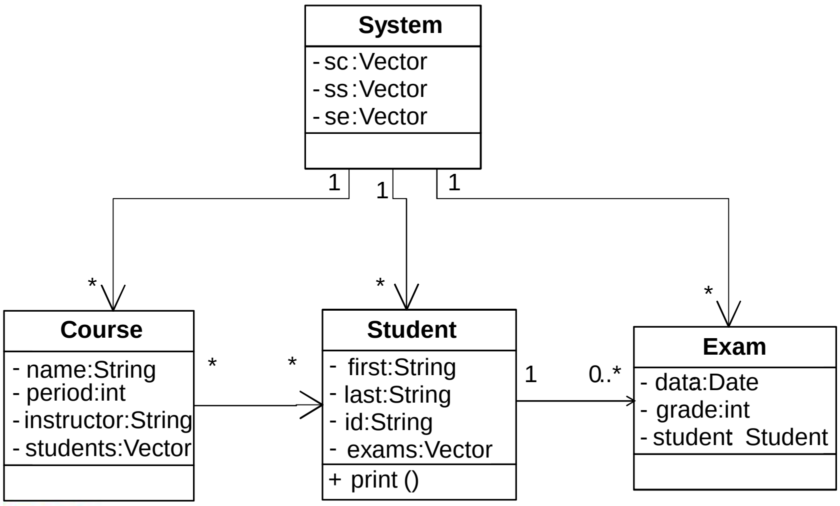
UML Package Diagram



How to implement associations

SoftEng
http://softeng.polito.it

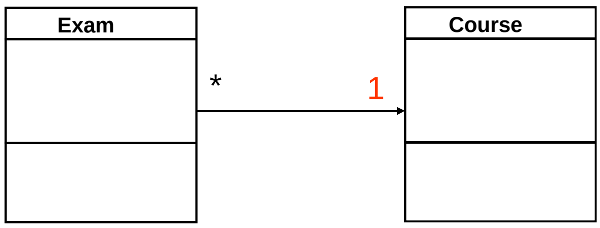
UML low-level design



SoftEng
http://softeng.polito.it

Association :1

- From Exam towards Course



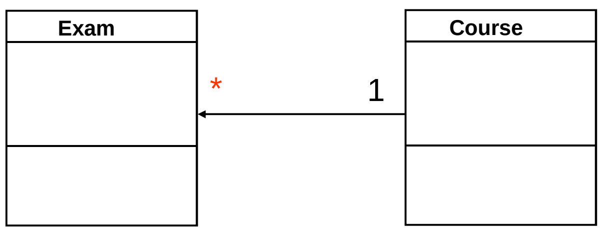

```
Class Exam {
  Course c;
  setCourse(Course c){
    this.c=c;}
}
```

```
Class Course {

}
```

Association :n

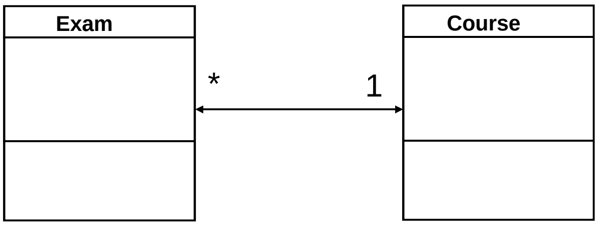
- From Course towards Exams




```
Class Course {
  ArrayList exams;
  Course(){ exams = new ArrayList (); }
  addExam(Exam e){ exams.add(e);}
}
```

Association 1:n

- Both directions



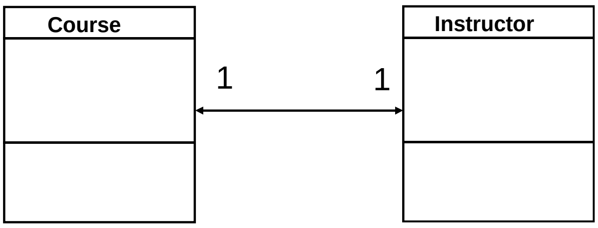

```
Class Exam {
  Course c;
  setCourse(Course c){
    this.c=c;
  }
}
```

```
Class Course {
 ArrayList exams;
 Course(){ exams = new ArrayList ();
}
  addExam(Exam e){ exams.add(e);}
}
```

SoftEng

Association 1:1

- Both directions



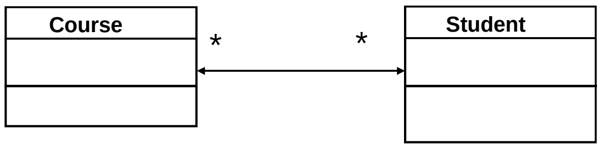

```
Class Course {
   Instructor i;
}
```

```
Class Instructor  {
  Course c;
}
```

SoftEng

Association n:m

- Both directions




```
Class Course {
  ArrayList students;
  Course(){
     students = new Vector();
  }
  addStudent(Student s){
    students.add(s);
  }
}
```

```
Class Student {
  ArrayList courses;
  Students(){
    courses = new Vector();
  }
  addCourse(Course c){
    courses.add(c);
  }
}
```

Summary – diagrams

- Static/structural view
  - Class diagram
- Functional view
  - Use Case diagram
- Dynamic view
  - Sequence diagram
  - Statechart & Activity Diagrams
- Physical view
  - Component & Deployment diagrams

Uses of UML [Fowler]

- Sketch
  - Used informally to share/discuss ideas
  - On whiteboard/paper
  - Meant to change
- Blueprint
  - Used in normative way to describe system to be built
  - On documents
  - Meant not to change
- Programming language
  - Model driven architecture
  - Forward and backward automatic transformations

SoftEng http://softeng.polito.it

Uses of UML in process [Fowler]

- Requirements
  - Use case diagrams, Class diagrams, activity diagrams, state diagrams
- Design
  - Class, sequence, package, state, deployment

SoftEng http://softeng.polito.it

Uses of UML in process

- In this course
- User requirements
  - Use cases, (activity diagrams)
- Developer requirements
  - Class diagrams, sequence diagrams
- Design
  - Class, deployment, package, statecharts.