SOFTWARE ENGINEERING

Week 1
Introduction & UML In a Nutshell

Prof. Dr. Muhittin GÖKMEN   Yard. Doç. Dr. A. Cüneyd TANTUĞ   Araş. Gör. Dr. Tolga OVATMAN

Istanbul Technical University
Computer Engineering Department

Course - Introduction

ଊ BLG411E – Software Engineering
- İTÜ Credits   : 3-0-0
- ECTS Credits : X
- Course Coordinator : Dr. A. Cüneyd TANTUĞ
- Course Web Site : www.ninova.itu.edu.tr
  - Lecture notes, announcements, report templates and examples, tools, etc.

2011-2012 Fall Semester		
	CRN12586	CRN12585
Instructor	Assist. Prof. Dr. A. Cüneyd TANTUĞ www.tantug.com	Prof. Dr. Muhittin GÖKMEN www.itu.edu.tr/gokmen
Location	4104	4316
Time	Tuesday, 13:30-16:30	Tuesday, 13:30-16:30
Teaching Assistant	Dr. Tolga OVATMAN www.itu.edu.tr/ovatman	

Course - Textbooks

Software Engineering:
A Practitioner's Approach (SEPA)
Roger S. Pressman, 6th ed.
McGraw-Hill, 2005

Software Engineering

Ian Sommerville
Addison-Wesley, 2010

Course - Supplementaries

Other Books	• Object-Oriented Software Engineering Using UML, Patterns and Java *Bernd Bruegge, Alan H. Dutoit, 3rd ed., Prentice Hall, 2009* • Yazılım Mühendisliği *Erhan Sarıdoğan, 1st ed., Papatya Yayıncılık, 2004*
Journals	• IEEE Transactions on Software Engineering • IEEE Software • ACM Transactions on Software Engineering and Methodology *Please click here for a larger list of journals.*
Societies	• The IEEE Computer Society
Other Links	• The Software Engineering Institute

Course - Outline

Week	Topic	Reading
1	**Introduction and UML in a Nutshell** Software Engineering Overview, Use Cases,Class Diagrams,Interaction Diagrams,State Chart Diagrams,Component Diagrams,Deployment Diagrams	
2	**Software Processes and Process Models** Process, Process Models, Waterfall Model, Iterative Models, Spiral Models, Unified Process	[SEPA1], [S01]
3	**Agile Software Development** Agile Methods, Extreme Programming, Scrum, Crystal *Recitation 1: A Broad view of software project development standards tools and processes*	[SEPA2], [S02]
4	**Software Project Management – 1** Management Spectrum, Project and Process Metrics, Estimation, Planning and Scheduling	[SEPA21,22,23,24], [SO22,23]
5	**Software Project Management – 2** Risk Management, Quality Management, Configuration/Change Management *Recitation 2: Configuration Management and software repositories using SVN, Requirements engineering using DOORS*	[SEPA25,26,27], [SO24,25,26]
6	**System Modeling and Requirements Engineering** System Engineering, Functional and Non-Functional Requirements, Requirements Specification, Elicitation and Analysis of Requirements, Validation of Requirements *Recitation 2: Configuration Management and software repositories using SVN, Requirements engineering using DOORS*	[SEPA7], [S04]
7	**Midterm Examination**	
8	**Analysis Model** Context Models, Interaction Models, Structural Models, Behavioural Models, Model-Driven Engineering *Recitation 3: Software design tools and Model driven engineering using GMF*	[SEPA6,7], [S05]
9	**Design Engineering – 1** Architectural Design, Component Level Design *Recitation 3: Software design tools and Model driven engineering using GMF*	[SEPA9,10], [S06]
10	**Design Engineering – 2** Model-View-Controller, Design Patterns, Implementation Issues, User Interface Design *Recitation 4: Software Development Utilities (Hibernate-LOG4J-DOM4J)*	[SEPA11,12], [S07]
11	**Software Testing** Testing Strategies, Testing Tactics, Test-Driven Development *Recitation 4: Software Development Utilities (Hibernate-LOG4J-DOM4J)*	[SEPA13,14], [S08]
12	**Web Engineering** Planning, Analysis, Design and Test for Web Applications, Project Management Issues for Web Applications *Recitation 5: Test Driven Software Development using JUnit and Mockito*	[SEPA16,17,18,19,20]
13	**Advanced Software Engineering** Service Oriented Architecture, Aspect Oriented Software Development, Formal Methods, Reengineering *Recitation 5: Test Driven Software Development using JUnit and Mockito*	[SEPA28,31], [S019,20,21]
14	**Project Presentations**	

Course - Grading

Midterm Exam (x 1)	%25
Final Exam	%40
Project	%35
Total	%100

Tentative (subject to change)

Course - Projects

- Large Team Projects
  - 20~25 people
  - One topic for all teams
    - 2010-2011 Fall Topic : University Student Automation System
- Team Organizations
  - Project Manager (will gain %5 more grade)
  - Requirement&Analysis Group
  - Design Group
  - Implementation Group
  - Testing Group
  - Project Related Activities
    - Quality Assurance
    - Change Manager
    - Documentation Manager

Introduction & UML   1.7

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
   - Package Diagrams
   - Deployment Diagrams

Introduction
୫୦ 1.1 ୯ଓ

Introduction & UML

What is Software Engineering?

- Formal Definition
  - The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"
    [IEEE Standard, 610.12, 1990].

Introduction & UML   1.9

What is Software Engineering?

- The study of systematic and effective processes and technologies for supporting software development and maintenance activities
  - Improve quality
  - Reduce costs
- Why is software engineering needed?
  - To predict time, effort, and cost
  - To improve software quality
  - To improve maintainability
  - To meet increasing demands
  - To lower software costs
  - To successfully build large, complex software systems
  - To facilitate group effort in developing software

Introduction & UML   1.10

Size of programs continues to grow...

- Trivial: 1 month, 1 programmer, 500 LOC,
  - Intro programming assignments
- Very small: 4 months, 1 programmer, 2000 LOC
  - Course project
- Small: 2 years, 3 programmers, 50K LOC
  - Nuclear power plant, pace maker
- Medium: 3 years, 10s of programmers, 100K LOC
  - Optimizing compiler
- Large: 5 years, 100s of programmers, 1M LOC
  - MS Word, Excel
- Very large: 10 years, 1000s of programmers, 10M LOC
  - Air traffic control,
  - Telecommunications, space shuttle
- Unbelievable: ? years, ? programmers
  - W2K 35M LOC
  - Missile Defense System 100M LOC?
  - Skynet ???

Introduction & UML   1.11

What's the problem?

- Software cannot be built fast enough to keep up with
  - H/W advances
  - Rising expectations
  - Feature explosion
- Increasing need for high reliability software
- Software is difficult to maintain
  - "aging software"
- Difficult to estimate software costs and schedules
- Too many projects fail
  - Ariane Missile
  - Denver Airport Baggage System
  - Therac

Introduction & UML   1.12

Software Disaster Examples - 1

Therac-25 (1985)

- **Cost:** Three people dead, three people critically injured
- **Disaster:** Canada's Therac-25 radiation therapy machine malfunctioned and delivered lethal radiation doses to patients.
- **Cause:** Because of a subtle bug called a race condition, a technician could accidentally configure Therac-25 so the electron beam would fire in high-power mode without the proper patient shielding.

Introduction & UML   1.13

Software Disaster Examples -2

Patriot Missile (1991)

- **Cost:** 28 soldiers dead, 100 injured
- **Disaster:** During the first Gulf War, an American Patriot Missile system in Saudi Arabia failed to intercept an incoming Iraqi Scud missile. The missile destroyed an American Army barracks.
- **Cause:** A software rounding error incorrectly calculated the time, causing the Patriot system to ignore the incoming Scud missile.

Introduction & UML   1.14

Software Disaster Examples - 3

Ariane 5 Rocket (1996)

- **Cost:** $500 million
- **Disaster:** Ariane 5, Europe's newest unmanned rocket, was intentionally destroyed seconds after launch on its first flight. Also destroyed was its cargo of four scientific satellites to study how the Earth's magnetic field interacts with solar winds.
- **Cause:** Shutdown occurred when the guidance computer tried to convert the sideways rocket velocity from 64-bits to a 16-bit format. The number was too big, and an overflow error resulted. When the guidance system shut down, control passed to an identical redundant unit, which also failed because it was running the same algorithm.

Introduction & UML   1.15
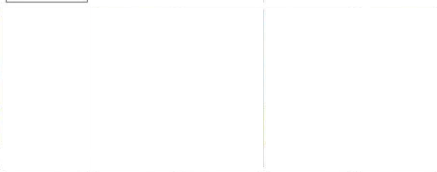
Why is software development so difficult?

- Communication
  - Between customer and developer
    - Poor problem definition is largest cause of failed software projects
  - Within development team
    - More people = more communication
    - New programmers need training

- Project characteristics
  - Novelty
  - Changing requirements
    - 5 x cost during development
    - up to 100 x cost during maintenance
  - Hardware/software configuration
  - Security requirements
  - Real time requirements
  - Reliability requirements

Introduction & UML   1.16

Communication Difficulties Example

How the customer explained it

Introduction & UML   17

Why is software development so difficult?

- Personnel characteristics
  - Ability
  - Prior experience
  - Communication skills
  - Team cooperation
  - Training
- Facilities and resources
  - Identification
  - Acquisition

- Management issues
  - Realistic goals
  - Cost estimation
  - Scheduling
  - Resource allocation
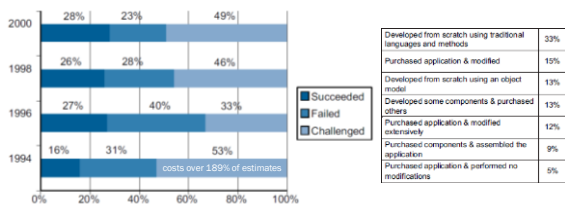 - Quality assurance
  - Version control
  - Contracts

Introduction & UML   18

3

Project Resolution History

Project Resolution History (1994–2000)

Developed from scratch using traditional languages and methods	33%
Purchased application & modified	15%
Developed from scratch using an object model	13%
Developed some components & purchased others	13%
Purchased application & modified extensively	12%
Purchased components & assembled the application	9%
Purchased application & performed no modifications	5%

Successful: The project is completed on time and on budget, with all features and functions originally specified.
Challenged: The project is completed and operational, but over-budget, over the time estimate, and with fewer features and functions than initially specified.
Failed: The project is cancelled before completion or never implemented

Source: Standish Group Chaos Report (survey on 352 companies, 2001)   1.19

Overrunning??

Cost Overruns	% of Responses
Under 20%	15.5%
21 – 50%	31.5%
51 – 100%	29.6%
101 – 200%	10.2%
201 – 400%	8.8%
Over 400%	4.4%

Time Overruns	% of Responses
Under 20%	13.9%
21 – 50%	18.3%
51 – 100%	20.0%
101 – 200%	35.5%
201 – 400%	11.2%
Over 400%	1.1%

% of Features/Functions	% of Responses
Less Than 25%	4.6%
25 – 49%	27.2%
50 – 74%	21.8%
75 – 99%	39.1%
100%	7.3%

Cost Overrun Data | Time Overrun Data | # of Feature Dropped

Factors Making SD Difficult

Project Challenged Factors	% of Responses
1. Lack of User Input	12.8%
2. Incomplete Requirements & Specifications	12.3%
3. Changing Requirements & Specifications	11.8%
4. Lack of Executive Support	7.5%
5. Technology Incompetence	7.0%
6. Lack of Resources	6.4%
7. Unrealistic Expectations	5.9%
8. Unclear Objectives	5.3%
9. Unrealistic Time Frames	4.3%
10. New Technology	3.7%
Other	23.0%

Factors Making SD Fail

Project Impaired Factors	% of Responses
1. Incomplete Requirements	13.1%
2. Lack of User Involvement	12.4%
3. Lack of Resources	10.6%
4. Unrealistic Expectations	9.9%
5. Lack of Executive Support	9.3%
6. Changing Requirements & Specifications	8.7%
7. Lack of Planning	8.1%
8. Didn't Need It Any Longer	7.5%
9. Lack of IT Management	6.2%
10. Technology Illiteracy	4.3%
Other	9.9%

Source: Standish Group   Introduction & UML   1.20

The Real Cost of Software Defects
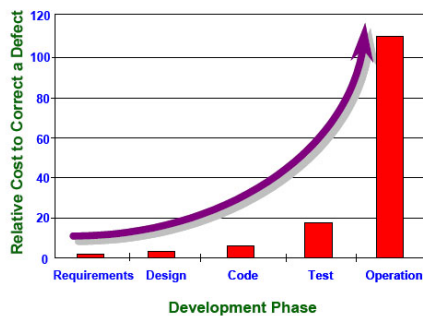
Introduction & UML   1.21

Some facts about software industry

- Cost per Line of Code
  $50 to $400 (Standish Group,1983)

- Errors found in code during development
  50-60 per KLOC

- Errors found in delivered code (defects)
  < 4 per KLOC (still a lot!)

- The US Department of Commerce estimates that software defects cost the US economy $60 Billion per year

- «In the last year, 70% of projects failed to meet deadlines, and 50% of projects failed to meet the needs of the business»
  «80% of the issues stem from poor requirements.»
  (Standish Group Chaos Report, 2007)

Introduction & UML   22

---

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
   - Package Diagrams
   - Deployment Diagrams

UML In A Nutshell

ಬ 1.2 ಜ

Introduction & UML

What is UML??

- Unified Modeling Language (UML) is the standard tool for visualizing, specifying, constructing, and documenting the artifacts of an object-oriented software.

- UML is not a programming language, but only a visual design notation.

- Can be used with all software development process models.

- Independent of implementation language.

- Many CASE tools uses UML for automatic code generation. Examples: IBM Rational, ArgoUML, etc.

- You may be familiar with some UML concepts introduced in Object Oriented Programming course.

Introduction & UML   24

Background

- UML is the result of an effort to simplify and consolidate the large number of **Object Oriented** development methods and notations.

- Developed by the Object Management Group based on work from:
  - Grady Booch [91]
  - James Rumbaugh [91]
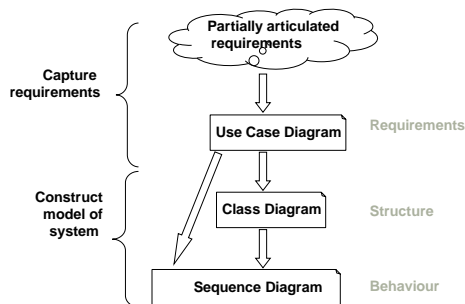 - Ivar Jacobson [92]

- The latest version is UML 2.0
  (See http://www.omg.org or http://www.uml.org)

Introduction & UML   1.25

Types of UML Diagrams

- **Use Case Diagrams (*)**

- **Class Diagrams (*)**

- Interaction Diagrams
  - **Sequence Diagrams (*)**
  - Collaboration Diagrams

- State Transition Diagrams

- Activity Diagrams

- Implementation Diagrams
  - Component Diagrams
  - Deployment Diagrams

Introduction & UML   1.26
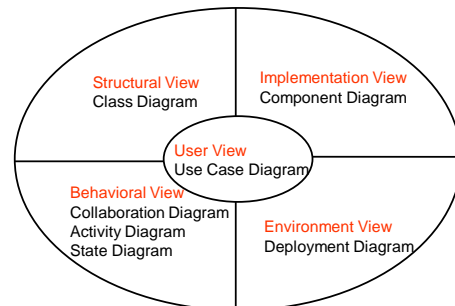
Minimal UML Process

Capture requirements
- Partially articulated requirements
- Use Case Diagram — Requirements

Construct model of system
- Class Diagram — Structure
- Sequence Diagram — Behaviour

Introduction & UML   1.27

Views of UML Diagrams

- Structural View — Class Diagram
- Implementation View — Component Diagram
- User View — Use Case Diagram
- Behavioral View — Collaboration Diagram, Activity Diagram, State Diagram
- Environment View — Deployment Diagram

Introduction & UML   1.28

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
   - Package Diagrams
   - Deployment Diagrams

UML In A Nutshell

29

USE CASES

Introduction & UML

Use Cases

- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- Main authors: *Booch, Rumbaugh, and Jacobson*
- The Object Management Group (OMG) is responsible for standardization. (www.omg.org)
- Current version is UML 2.0

Introduction & UML   1.30

Use Case : Actors

- ✦ An actor represents a set of roles that users of use case play when interacting with these use cases.
  - ○ Actors can be human or automated systems.
  - ○ Actors are entities which require help from the system to perform their task or are needed to execute the system's functions.
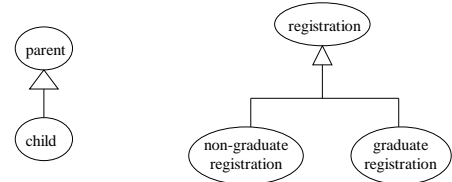 - ○ Actors are not part of the system.

name
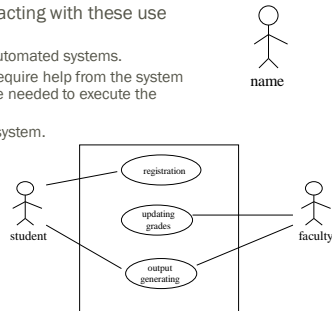
Introduction & UML   1.31

Uses Case Relationships : Generalization

1. Generalization - use cases that are specialized versions of other use cases.
  - ○ The child use case inherits the behavior and meaning of the parent use case.
  - ○ The child may add to or override the behavior of its parent.

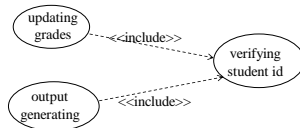Introduction & UML   1.32

Uses Case Relationships : Include

2. Include - use cases that are included as parts of other use cases. Enable to factor common behavior.
  - ○ The base use case explicitly incorporates the behavior of another use case at a location specified in the base.
  - ○ The included use case never stands alone. It only occurs as a part of some larger base that includes it.
  - ○ Enables to avoid describing the same flow of events several times by putting the common behavior in a use case of its own
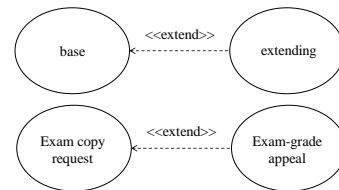

Introduction & UML   1.33

Uses Case Relationships : Extend

3. Extend - use cases that extend the behavior of other core use cases. Enable to factor variants.
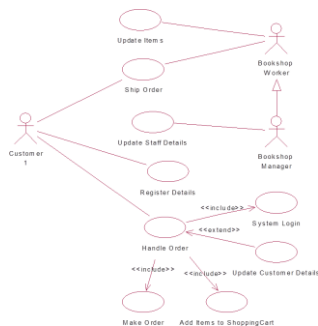 - ○ The base use case implicitly incorporates the behavior of another use case at certain points called extension points.
  - ○ The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case.
  - ○ Enables to model optional behavior or branching under conditions.

Introduction & UML   1.34

Example : Use Case Diagram

Introduction & UML   1.35

Use Case Description

Each use case may include all or part of the following:

▪ Title or Reference Name	- meaningful name of the UC
▪ Author/Date	- the author and creation date
▪ Modification/Date	- last modification and its date
▪ Purpose	- specifies the goal to be achieved
▪ Overview	- short description of the processes
▪ Cross References	- requirements references
▪ Actors	- agents participating
▪ Pre Conditions	- must be true to allow execution
▪ Post Conditions	- will be set when completes normally
▪ Normal flow of events	- regular flow of activities
▪ Alternative flow of events	- other flow of activities
▪ Exceptional flow of events	- unusual situations
▪ Implementation issues	- foreseen implementation problems

Introduction & UML   1.36

Example : Use Case (Money Withdraw) - I

- **Use Case:** Withdraw Money
- **Author:** ZB
- **Date:** 1-OCT-2004
- **Purpose:** To withdraw some cash from user's bank account
- **Overview:** The use case starts when the customer inserts his credit card into the system. The system requests the user PIN. The system validates the PIN. If the validation succeeded, the customer can choose the withdraw operation else alternative 1 – validation failure is executed. The customer enters the amount of cash to withdraw. The system checks the amount of cash in the user account, its credit limit. If the withdraw amount in the range between the current amount + credit limit the system dispense the cash and prints a withdraw receipt, else alternative 2 – amount exceeded is executed.
- **Cross References:** R1.1, R1.2, R7

Introduction & UML 1.37

Example : Use Case (Money Withdraw) - II

- **Actors:** Customer
- **Pre Condition:**
  - The ATM must be in a state ready to accept transactions
  - The ATM must have at least some cash on hand that it can dispense
  - The ATM must have enough paper to print a receipt for at least one transaction
- **Post Condition:**
  - The current amount of cash in the user account is the amount before the withdraw minus the withdraw amount
  - A receipt was printed on the withdraw amount
  - The withdraw transaction was audit in the System log file

Introduction & UML 1.38

Example : Use Case (Money Withdraw) - III

Typical Course of events:

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

Introduction & UML 1.39

Example : Use Case (Money Withdraw) - IV

- **Alternative flow of events:**
  - Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - Step 8: Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - Step 8: Customer exceeds its legal amount. Display an error message, and go to step 6.
- **Exceptional flow of events:**
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card

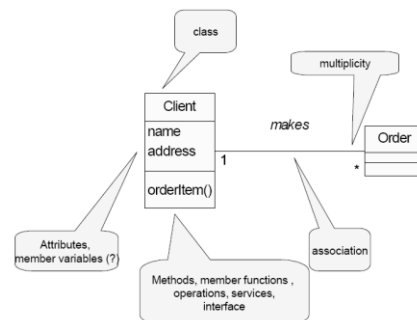Introduction & UML 1.40

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
   - Package Diagrams
   - Deployment Diagrams

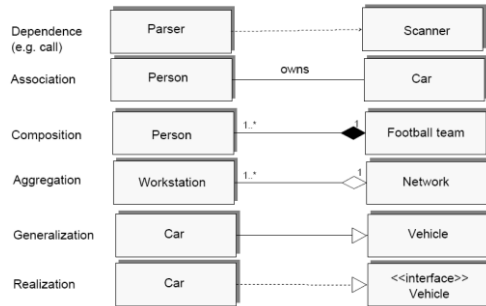# UML In A Nutshell

∞ 41 ∞

CLASS DIAGRAMS

Introduction & UML

Class Diagrams

Introduction & UML 1.42
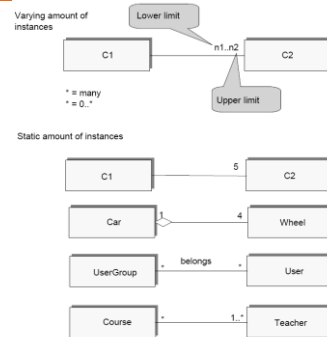
7

Class Relations

Dependence (e.g. call)	Parser	Scanner
Association	Person — owns — Car	
Composition	Person 1..* ◆—1 Football team	
Aggregation	Workstation 1..* ◇—1 Network	
Generalization	Car —▷ Vehicle	
Realization	Car --▷ <<interface>> Vehicle	

Introduction & UML   1.43

Cardinality

Varying amount of instances

Lower limit

C1 ——n1..n2—— C2

Upper limit

* = many
* = 0..*

Static amount of instances

C1 ——5—— C2

Car 1——4 Wheel

UserGroup — belongs — User

Course * ——1..* Teacher

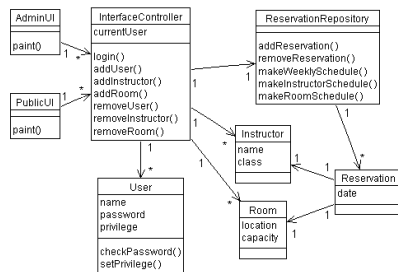Introduction & UML   1.44

Example : Class Diagram
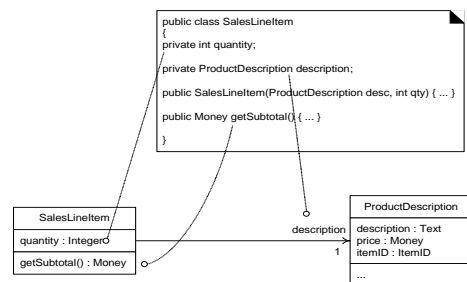
A classroom scheduling system: specification perspective.

Introduction & UML   1.45

Example : From Class Diagram to Code

```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

SalesLineItem
quantity : Integer
getSubtotal() : Money

description

1

ProductDescription
description : Text
price : Money
itemID : ItemID
...

Introduction & UML   1.46

1. Course Objectives
2. UML In A Nutshell
   • Use Cases
   • Class Diagrams
   • Interaction Diagrams ⇐
   • State Chart Diagrams
   • Package Diagrams
   • Deployment Diagrams

UML In A Nutshell

ဆ 47 03

INTERACTION DIAGRAMS

Introduction & UML

Interaction Diagrams
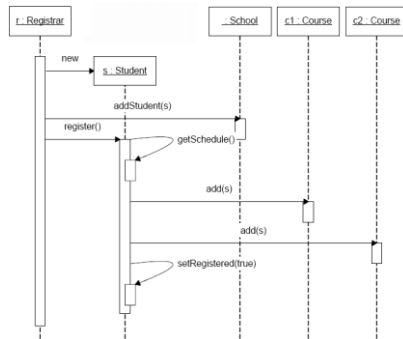
ဆ UML presents two types of interaction diagrams.
1. Sequence Diagrams
2. Collaboration Diagrams

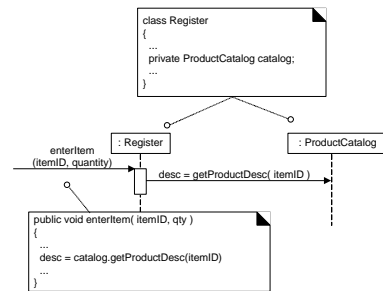Introduction & UML   48

Sequence Digrams

r : Registrar | : School | c1 : Course | c2 : Course

new → s : Student
addStudent(s)
register()
getSchedule()
add(s)
add(s)
setRegistered(true)

Introduction & UML   1.49

Example : Mapping Diagram to Code

Mapping sequence diagram to Java code


```
class Register
{
    ...
    private ProductCatalog catalog;
    ...
}
```

: Register   : ProductCatalog

enterItem
(itemID, quantity)

desc = getProductDesc(itemID)

```
public void enterItem( itemID, qty )
{
    ...
    desc = catalog.getProductDesc(itemID)
    ...
}
```

Introduction & UML   1.50

Collaboration Diagrams


```
{
    ProductDescription desc = catalog.ProductDescription(id);
    currentSale.makeLineItem(desc, qty);
}
```

enterItem(id, qty) → :Register → 2: makeLineItem(desc, qty) → :Sale

1: desc := getProductDescription(id)
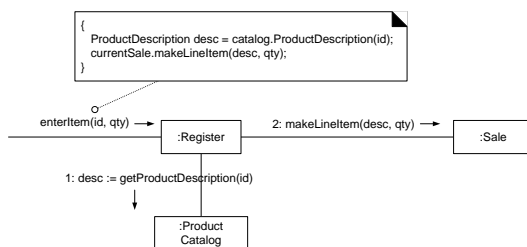
:Product
Catalog

Introduction & UML   1.51

1. Course Objectives
2. UML In A Nutshell
   • Use Cases
   • Class Diagrams
   • Interaction Diagrams
   • State Chart Diagrams ⇐
   • Package Diagrams
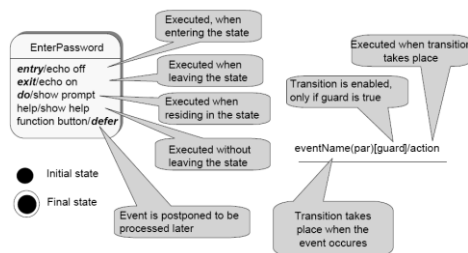   • Deployment Diagrams

UML In A Nutshell

೮೦ 52 ೦೪

STATE CHART DIAGRAMS
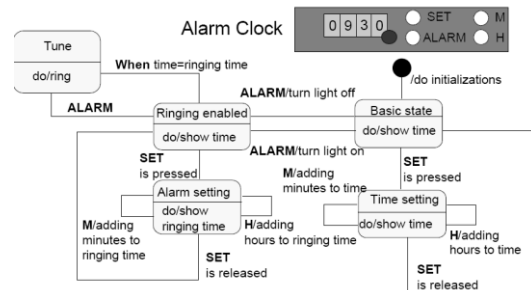
Introduction & UML

State Chart Notation

• State transition diagram shows
  ▪ The events that cause a transition from one state to another
  ▪ The actions that result from a state change

EnterPassword
entry/echo off
exit/echo on
do/show prompt
help/show help
function button/*defer*

Executed, when entering the state
Executed when leaving the state
Executed when residing in the state
Executed without leaving the state

Executed when transition takes place
Transition is enabled, only if guard is true

eventName(par)[guard]/action

● Initial state
◉ Final state

Event is postponed to be processed later
Transition takes place when the event occures

Introduction & UML   53

Example : State Chart Diagram

Alarm Clock   0 9 3 0   ● SET   ● M
                          ○ ALARM ○ H

Tune
do/ring

When time=ringing time
ALARM/turn light off
/do initializations
ALARM

Ringing enabled
do/show time

Basic state
do/show time

ALARM/turn light on

SET is pressed

M/adding minutes to time

SET is pressed

Alarm setting
do/show ringing time

Time setting
do/show time

M/adding minutes to ringing time

H/adding hours to ringing time

H/adding hours to time

SET is released

SET is released

Introduction & UML   1.54

9

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
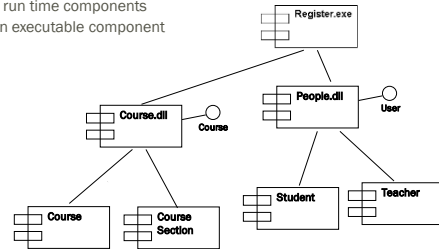   - Package Diagrams ←
   - Deployment Diagrams

UML IN A Nutshell

ℰ 55 ℭ

PACKAGE DIAGRAMS

Introduction & UML

Package Diagrams

- Component diagrams illustrate the organizations and dependencies among software components in physical world.
- A component may be
  - A source code component
  - A run time components
  - An executable component

Introduction & UML   56

---

1. Course Objectives
2. UML In A Nutshell
   - Use Cases
   - Class Diagrams
   - Interaction Diagrams
   - State Chart Diagrams
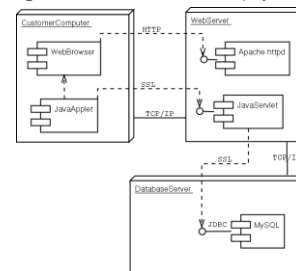   - Package Diagrams
   - Deployment Diagrams ←

UML In A Nutshell

ℰ 57 ℭ

DEPLOYMENT DIAGRAMS

Introduction & UML

Deployment Diagrams

- Captures the distinct number of computers involved
- Shows the communication modes employed
- Component diagrams can be embedded into deployment diagrams effectively

Introduction & UML   58