SOFT2201 introduction to software engineering and modelling

Week 1

**Agenda**

– **Software Engineering**

– **Why, What**

– **Software Modeling**

– **What, Why and How**

– **The Unified Process**

**Software engineering**

**Why**:

* Need to build **high-quality software** systems under **resource constraints**
* Social: user need, convenience
* Economical: cost
* Time: deliver of product

**What**:

* Accidents: difficulties related to the production of software
* Essences: difficulties inherent (or intrinsic) in the nature of software

• **Complexity**

• Many diverse software entities - interactions increase exponentially

• Intrinsic complexity cannot be abstracted - aircraft software, air traffic control

• **Conformity**

• Arbitrary changes from environment (people, systems) - no unifying principle

• **Changeability**

• Changing a building model vs. a software

• Stakeholders understanding of software changes

• **Invisibility**

• Software is intangible (invisible)

• Building model vs software models (UML - 13 diagram types)

* Engineers use Theories, methods, tools, techniques and approaches to manage the project, make good product
* Engineers process for managing and developing, account Dependability, Performance and costs.

Software Design/Modelling & Construction

**What**: Software Modelling

• The process of **developing conceptual models of a system at different levels of abstraction**

• Fulfil the defined **system requirements**

• Focus on important system **details to deal with complexity**

• **Object-oriented** design approach

• Concepts in the domain problem are modelled as interacting objects

• Using **graphical notations** (e.g., UML) to capture different views or perspectives

**WHY**: **Abstraction**

• Conceptual process to **derive general rules and concepts from concrete information in a certain context**

• Analysis and understanding of domain-specific problems

• **Decompose large problems into smaller** understandable piece

• software models with **different views/perspective**s

• Time, structural, requirements, etc.

**HOW: UML Principles**

• Graphical notations to visually specify and document design artifacts software systems using OO concepts

• Combines techniques from data modeling (ER diagrams), business modeling (workflows), object and component modeling

• Capture system activities (jobs), system components and its interaction, software entities and its interaction, run-time behavior, external interfaces

* + UML require design knowledge and thinking in objects

• Three ways to apply UML

• Sketching to explore part of the problem or solution space

• Blueprint: detailed design diagrams for:

• Reverse engineering to visualize and understand existing code

• Forward engineering (code generation)

• Programming language (Model Driven Engineering): executable specification of a system

**The Rational Unified Process(UP)**

* UP: an **iterative and Evolutionary** Development
* Agile: a practice of UP

Software Construction / Implementation

• Realization of design to produce a working software system

• Meet **customer requirements**

• Design and implementation activities often interleaved

• **Agile** development to **accommodate for changes**

• **Object-Oriented** design and Implementation model

• **Encapsulation**

• **Abstraction**

• **Reuse**

• **Maintenance**

Week 2

**Agenda**

– UML Modeling

– UML **Use Case** Diagrams

– UML **Class** Diagrams

– UML **Interaction** Diagrams

– Case Study: Next Gen Point-of-Sale (POS) System

– Use Cases

– Domain models

– Class and Sequence Diagrams

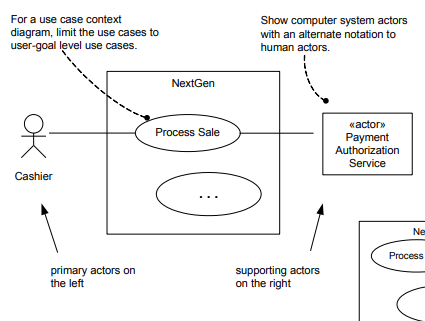
**Use case diagrams**

Subject: system under the use case.

Actor: role that interact with the subject/system

Association: relationship between an actor and a use case

«include»: the behavior of the included use case is included in the behavior of the including use case



**Class diagrams – structural diagrams(描述多个class和他们之间的联系)**

Perspectives:

* + **Conceptual**: describes key concepts in the problem domain. Use in business modeling for OO analysis(基础)
  + **Specification**: describes software components with specification and interfaces（进阶）
  + **Implementation**: describes software implementation in a particular programming language (e.g., Java)（包含语言）

Notations:

• Common compartments: classifier name, attributes and operations

• Package name

• <<interface>>

• Dependency

• Class hierarchy – inheritance

• Association and multiplicity

• Optional and default elements

Attributes:

+ 代表public, - 代表private, 没有默认private

{}大括号内代表附加属性, eg: {readOnly}

Operations:

* + declaration with a name, parameters, return type, exception list, and possibly a set of constraints of pre-and post-conditions.
  + 默认public
  + Eg. + getPlayer (name : String) : Player {exception IOException} (UML格式)

== Public Player getPlayer(String name) throws IOException(用编程语言做签名)

Methods:

用伪代码写出输入输出及process.

Stereotypes:

• **Stereotypes**: allow refinement (extension) of an existing modeling concept

• **UML profile**: group of related model elements allow customizing UML models for a specific domain or platform

Generalisation:

• Classes and operations with **{abstract}** tag are **abstract**

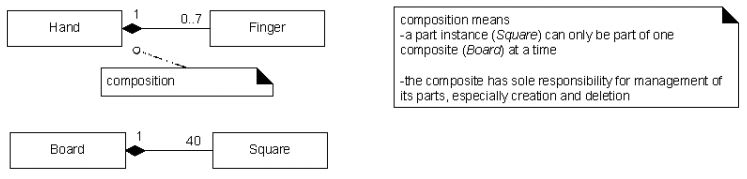
• Classes and operations with **{leaf}** are **final** (cannot be overridden in the sub-classes)

**Composition**, or composite aggregation, relationship:

• Instance of the **part** (e.g., Square) **belongs to only one composite** instance at a time (e.g., one board)

• The **part must always belong to a composite**

• The composite is responsible for the creation and deletion of its parts (by itself or by collaborating with other objects)



**Constraints**:

• needs to be described on the diagram by **quotations**.

**Associations**:

• Relationship between classifiers where **logical or physical** link exists among classifier’s instances

• **Notations**: • Association name (relation的名字) • Multiplicity(一对多关系)

Dependency

• A dependency exists between two elements if changes to the definition of one element (the supplier) may cause changes to the other (the client)

• There are many varieties of dependency, like **call, use, parameter**

**Interaction Diagrams- Dynamic (Behavioural) Diagrams**

**Sequence diagrams**: illustrate **sequence/timeordering** of messages in a **fence format** (each object is added to the right)

• Standard message syntax in UML :

Return = message (parameter : parameterType) : returnType

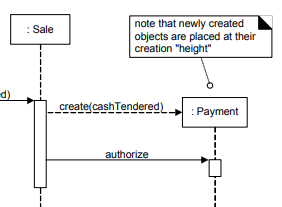
• Examples:

descrp = getProductDescription(id) : ProductDescription

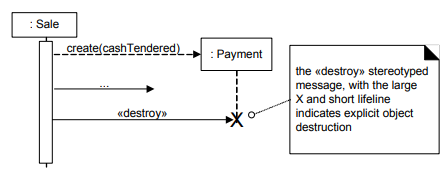
• Message reply/return

returnVar = message (parameter)

• Object Creation

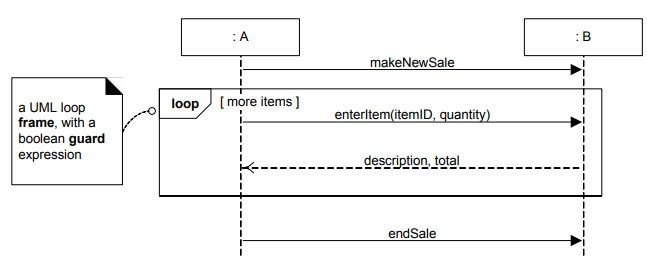


• Object Destruction



• Frames

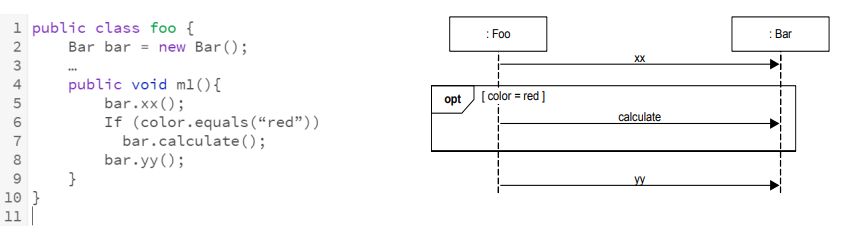
例：循环



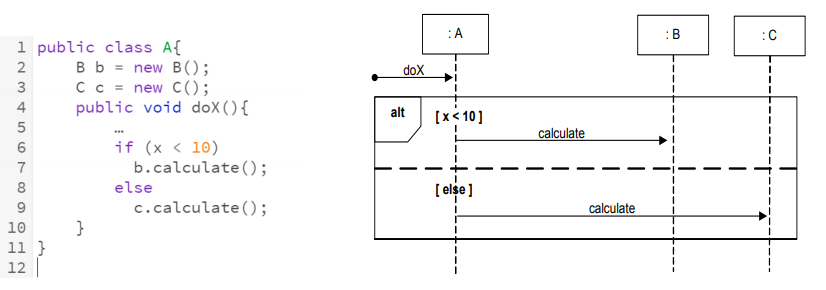
Common frame operators:

|  |  |
| --- | --- |
| Frame operator | Meaning |
| Alt(互异) | Alternative fragment for mutual exclusion conditional logic expressed in the guards |
| Loop（循环） | Loop fragment while guard is true |
| Opt（条件） | Optional fragment that executes if guard is true |
| Par（平行） | Parallel fragments that execute in parallel |
| Region（领域， 只有一个线程可运行） | Critical region within which only one thread can run |

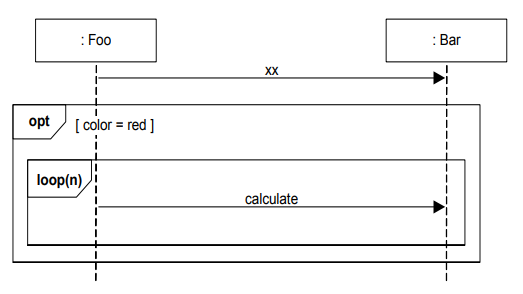
例：条件



例：互异



例：嵌套



**Software Construction / Implementation**

• Realization of design to produce a working software system

• Meet **customer requirements**

• Design and implementation activities often interleaved

• **Agile** development to **accommodate for changes**

• **Object-Oriented design** and Implementation model

**• Encapsulation**

**• Abstraction**

**• Reuse**

**• Maintenance**