ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

SYSTEMS ANALYSIS AND DESIGN

Lecturers: Nguyen Thanh Binh – Nguyen Quang Vu – Le Viet Truong – Le Thi Bich Tra – Vo Van Luong – Nguyen Thi Hanh

Faculty of Computer Science

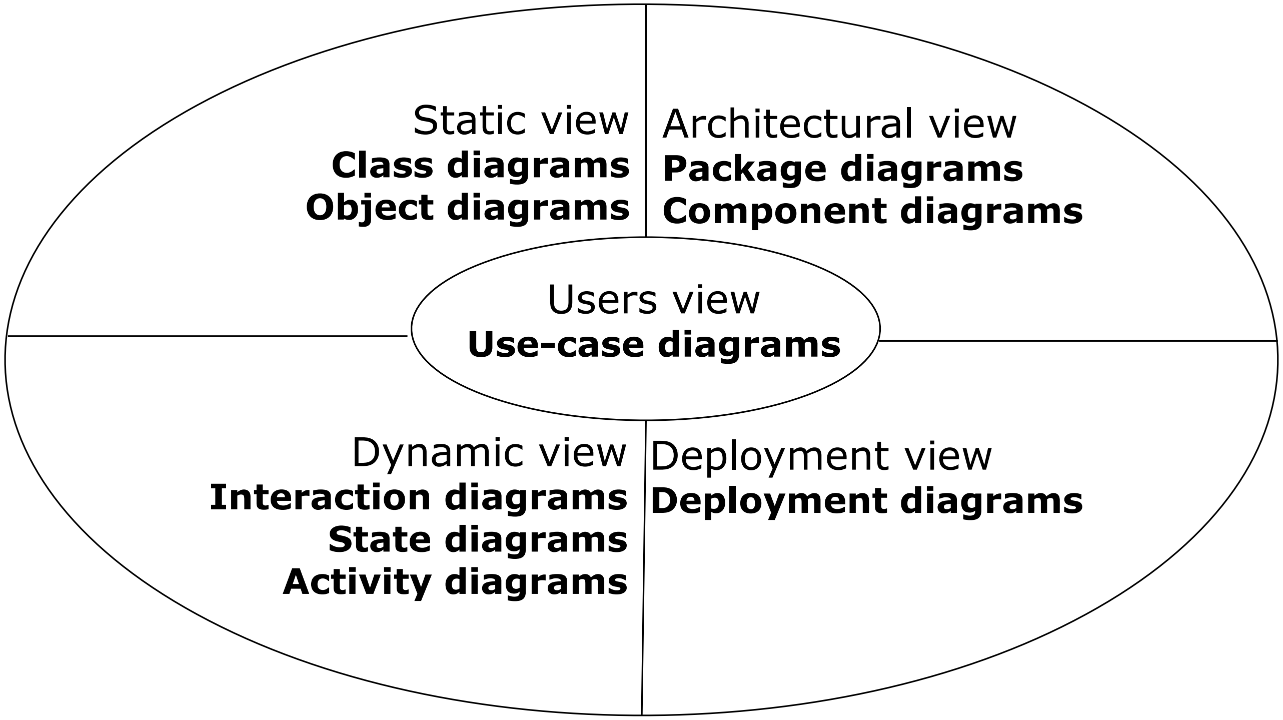
Vietnam - Korea University of Information and Communication Technology (VKU)

http://vku.udn.vn/

Modelling static structure

- Class diagrams
- Object diagrams

Views



Static view
Class diagrams
Object diagrams

Architectural view
Package diagrams
Component diagrams

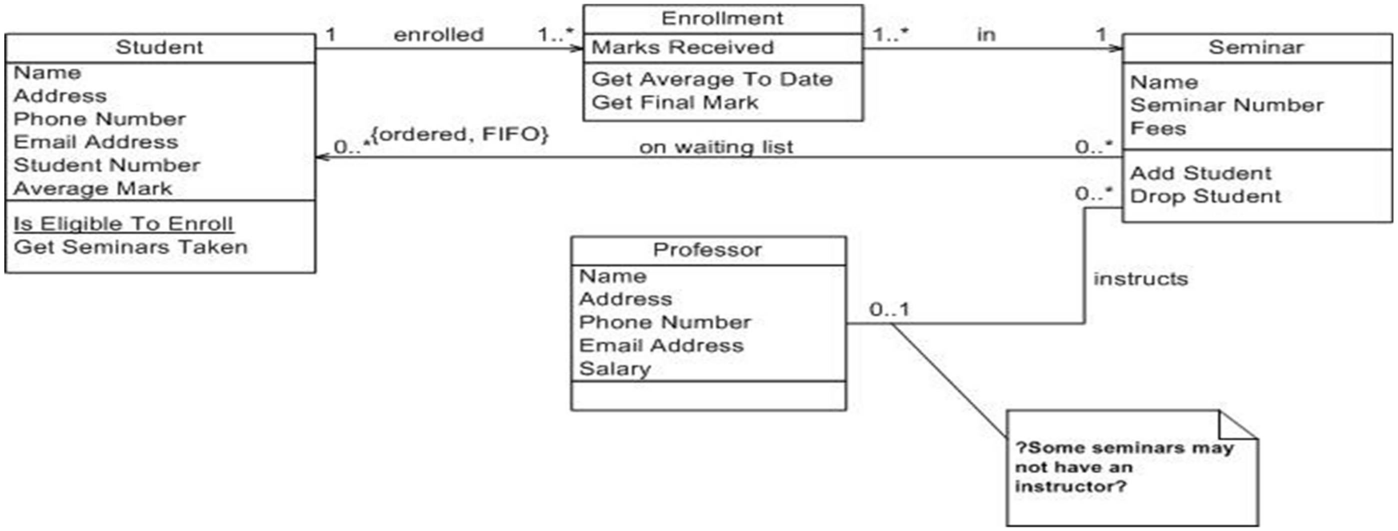
Users view
Use-case diagrams

Dynamic view
Interaction diagrams
State diagrams
Activity diagrams

Deployment view
Deployment diagrams

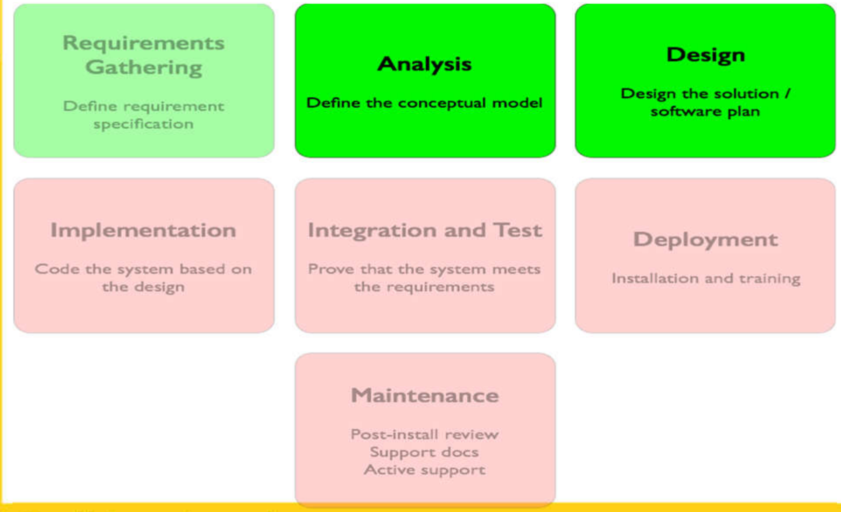
Class diagrams

- Class diagrams
  - consist of a set of classes, interfaces and their relationships
  - represent the **static view** of the system
  - can produce / build the **skeleton** of the system
- Modelling class diagrams is the **essential step** in object-oriented design



Analysis class diagram v.s Design class diagram

- Two main types of class diagrams
  - Conceptual/Analysis class diagram (domain model)
    - is developed in the analysis phase
    - describes the system from the “user point of view”
  - Design class diagram
    - is developed in the design phase basing
    - describes the system from the “software developer point of view”


Requirements Gathering
Define requirement specification

Analysis
Define the conceptual model

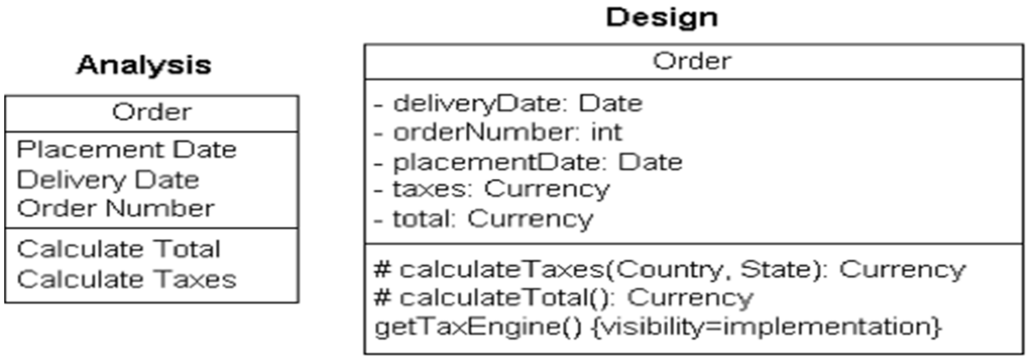
Design
Design the solution / software plan

Implementation
Code the system based on the design

Integration and Test
Prove that the system meets the requirements

Deployment
Installation and training

Maintenance
Post-install review
Support docs
Active support


Analysis

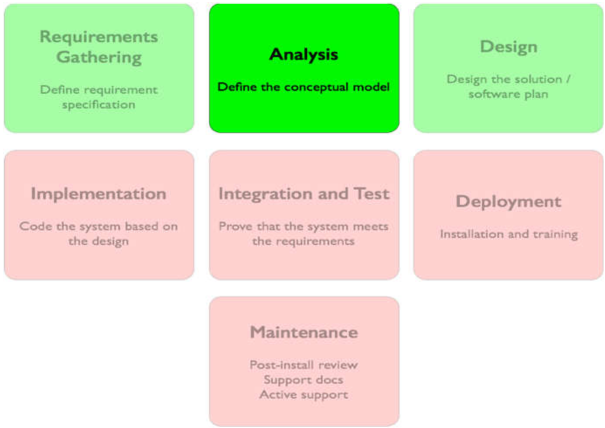
Order
Placement Date
Delivery Date
Order Number
Calculate Total
Calculate Taxes

Design

Order
- deliveryDate: Date
- orderNumber: int
- placementDate: Date
- taxes: Currency
- total: Currency
# calculateTaxes(Country, State): Currency
# calculateTotal(): Currency
getTaxEngine() {visibility=implementation}

Analysis Class Diagram

- Conceptual/analysis class diagram (domain model)
  - is constructed in the analysis phase
  - captures the concepts recognized by user/customer/stakeholder
  - doesn't contain information of how the software system should be implemented



Requirements Gathering
Define requirement specification

Analysis
Define the conceptual model

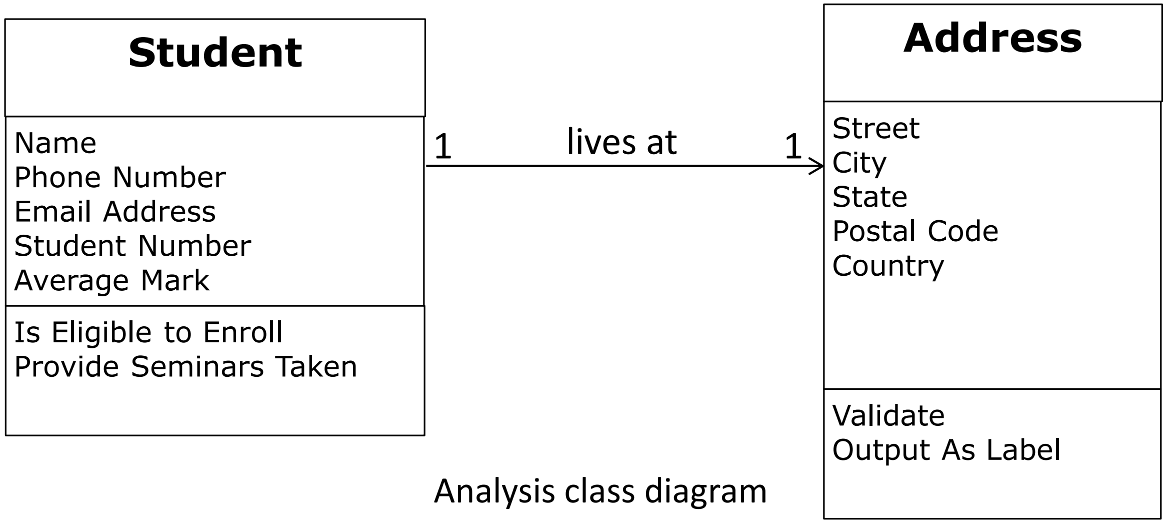
Design
Design the solution / software plan

Implementation
Code the system based on the design

Integration and Test
Prove that the system meets the requirements

Deployment
Installation and training

Maintenance
Post-install review
Support docs
Active support



Student

Name
Phone Number
Email Address
Student Number
Average Mark

Is Eligible to Enroll
Provide Seminars Taken

1 lives at 1

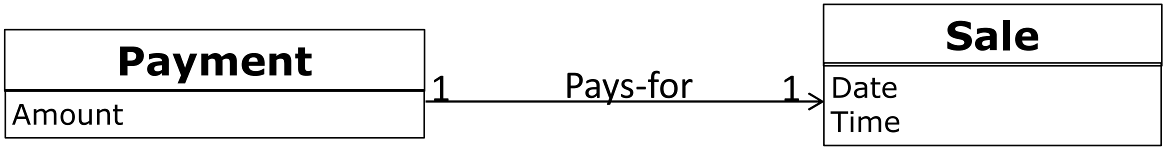
Address

Street
City
State
Postal Code
Country

Validate
Output As Label

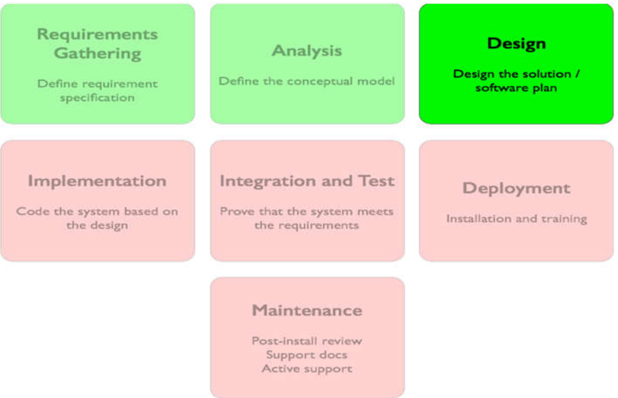
Analysis class diagram

- Another Example



Design Class Diagram

- Design class diagram
  - is construct in the design phase
  - a detail version of the analysis class diagram
    - an analysis class may correspond to several design classes
  - contains information about how the software system should be implemented
    - attributes’ and methods’ visibility
    - attributes’ and methods’ name conform to the target programming language


Requirements Gathering
Define requirement specification

Analysis
Define the conceptual model

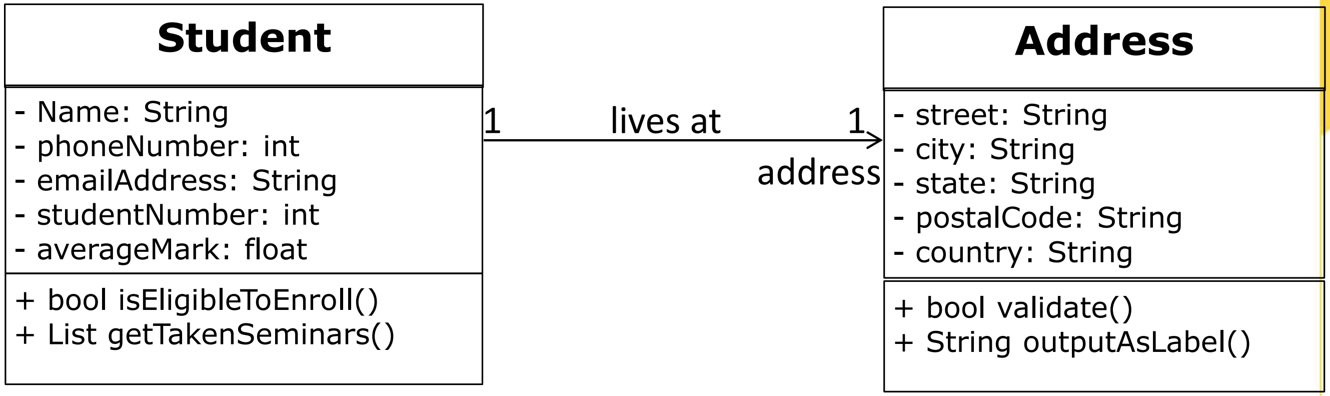
Design
Design the solution / software plan

Implementation
Code the system based on the design

Integration and Test
Prove that the system meets the requirements

Deployment
Installation and training

Maintenance
Post-install review
Support docs
Active support


Student

- Name: String
- phoneNumber: int
- emailAddress: String
- studentNumber: int
- averageMark: float

+ bool isEligibleToEnroll()
+ List getTakenSeminars()

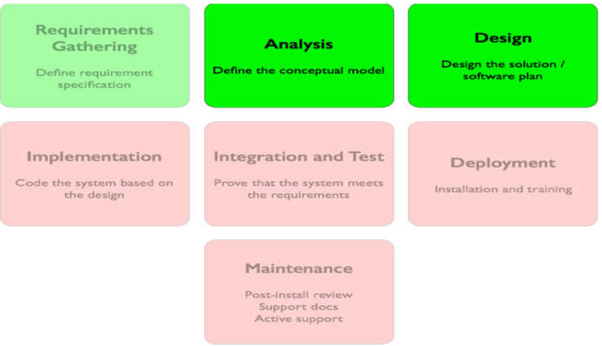
Address

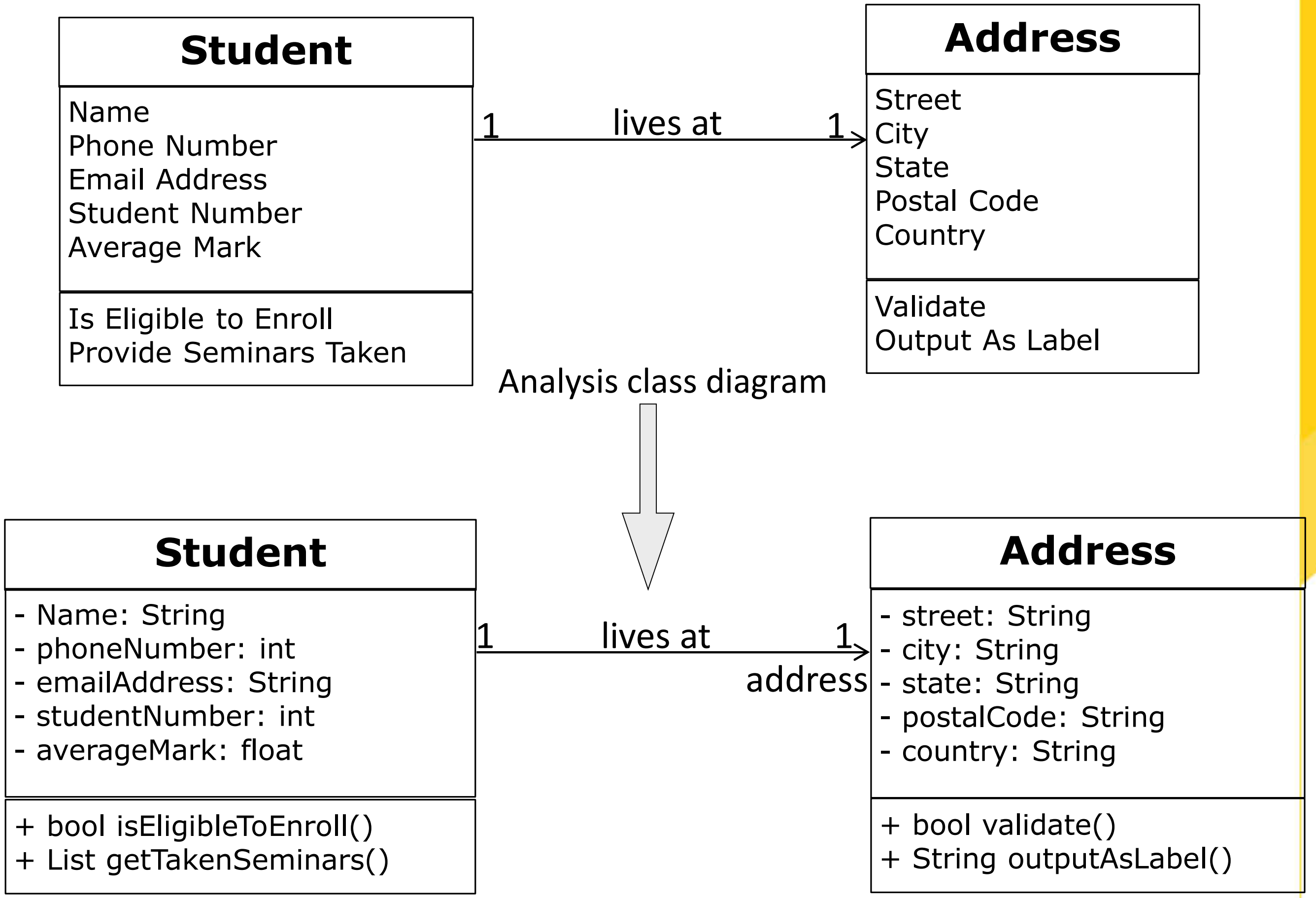
- street: String
- city: String
- state: String
- postalCode: String
- country: String

+ bool validate()
+ String outputAsLabel()

Design class diagram (for Java implementation)

Analysis Class Diagram v.s. Design Class Diagram


Requirements Gathering
Define requirement specification
Analysis
Define the conceptual model
Design
Design the solution / software plan
Implementation
Code the system based on the design
Integration and Test
Prove that the system meets the requirements
Deployment
Installation and training
Maintenance
Post-install review
Support docs
Active support


Student
Name
Phone Number
Email Address
Student Number
Average Mark
Is Eligible to Enroll
Provide Seminars Taken

1 lives at 1

Address
Street
City
State
Postal Code
Country
Validate
Output As Label

Analysis class diagram

Student
- Name: String
- phoneNumber: int
- emailAddress: String
- studentNumber: int
- averageMark: float

+ bool isEligibleToEnroll()
+ List getTakenSeminars()

1 lives at 1
address

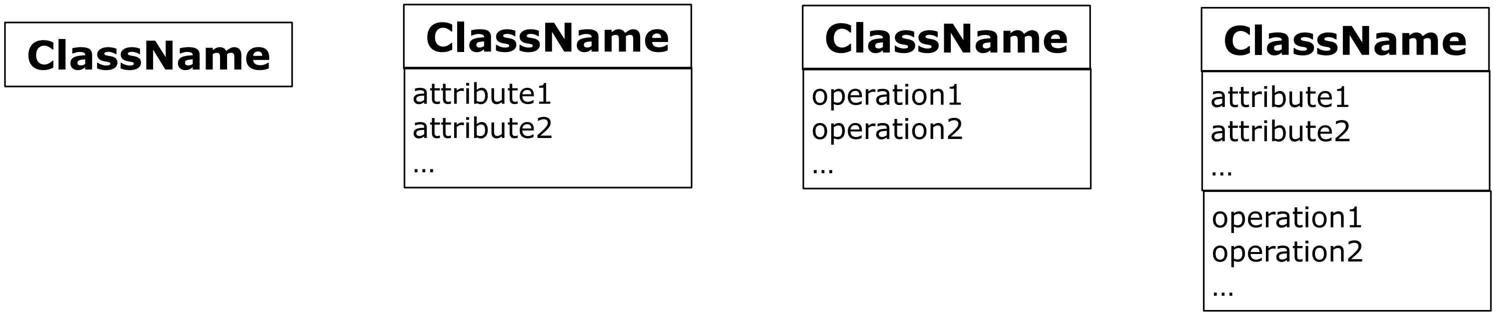
Address
- street: String
- city: String
- state: String
- postalCode: String
- country: String

+ bool validate()
+ String outputAsLabel()

Design class diagram (for Java implementation)

Class

- UML class
  - represents the class or interface concept of object-oriented programming language
  - consists of a set of attributes and operation
  - can be graphically represented in several forms

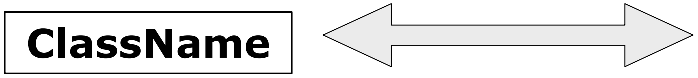


ClassName

ClassName
attribute1
attribute2
…

ClassName
operation1
operation2
…

ClassName
attribute1
attribute2
…
operation1
operation2
…



ClassName

```
class ClassName {
 …
};
```

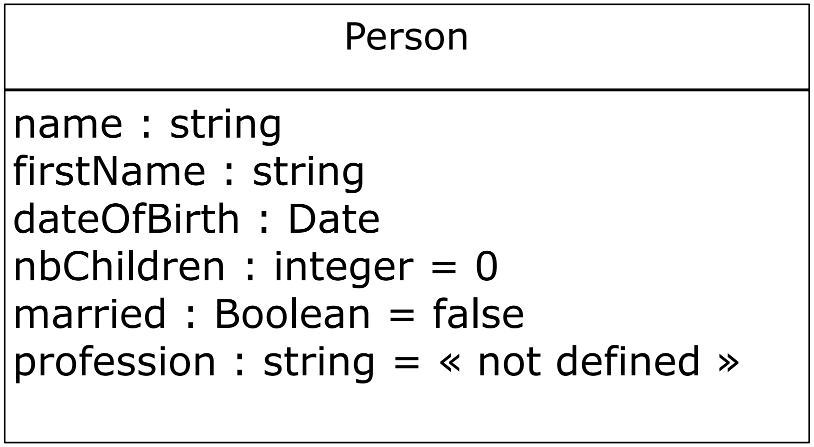
C++

```
public class ClassName {
 …
};
```

Java

Attributes

- Attributes represent the necessary data of class instances
- Attributes can have
  - a type
    - simple type
      - number : integer
      - length : double
      - text : string
    - complex type
      - center : Point
      - date : Data
  - A value by default
    - number : integer = 10
  - A list of possible value
    - color : Color = red {red, blue, purple, yellow}



Person

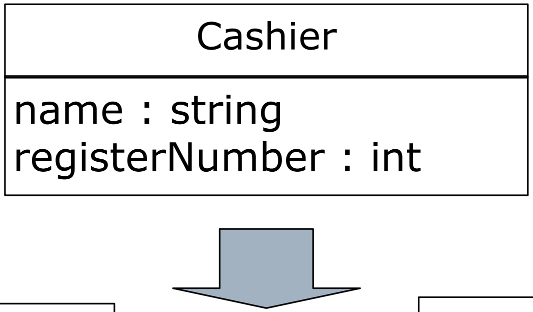
name : string
firstName : string
dateOfBirth : Date
nbChildren : integer = 0
married : Boolean = false
profession : string = « not defined »

VDA3

Attributes

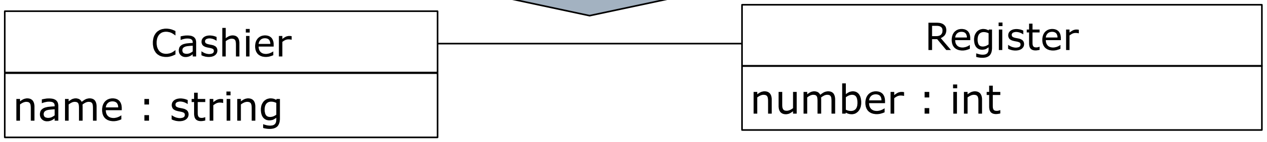
- An attribute represents only the data related to **the class that has this attribute**

- Example



Cashier

name : string
registerNumber : int



Cashier

name : string

Register

number : int

Slide 12

VDA3 move to “Developing Class Diagram”

VO Duc An, 29/06/2015

Attributes

- Identifying attributes
  - Attributes are **numbers** or **strings**
    - Since attributes represent the characteristics of the objects
  - Distinguishing between attributes and classes
    - If a characteristic of a class is not capable of doing something, then this is possibly an attribute
    - If we doubt that an attribute is a class, then it is considered as a class
      - Does that *salary* is an attribute of *Person* class?
      - If in doubt, then we consider that the *salary* is a class.

VDA4

Slide 13

VDA4 Identifying attributes from the (textual) specification

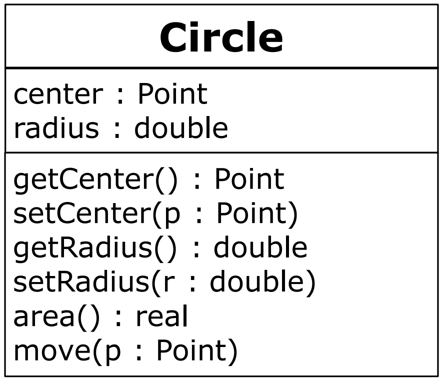
TODO an example?

VO Duc An, 17/06/2015

Operations / Methods

- Operations represent the **behaviors** of instance of the class
- The behavior of a class includes
  - The **getters** and **setters** that manipulate the data of class instances
  - A certain number of tasks associated with the **responsibility** of the class

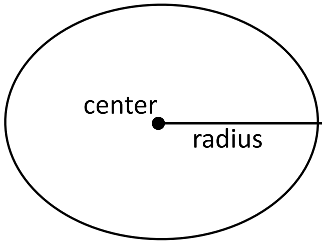
- Operations can have
  - a name
    - area, calculate, …
  - a returned type
    - area() : double
  - arguments with type
    - move(p : Point)



Circle

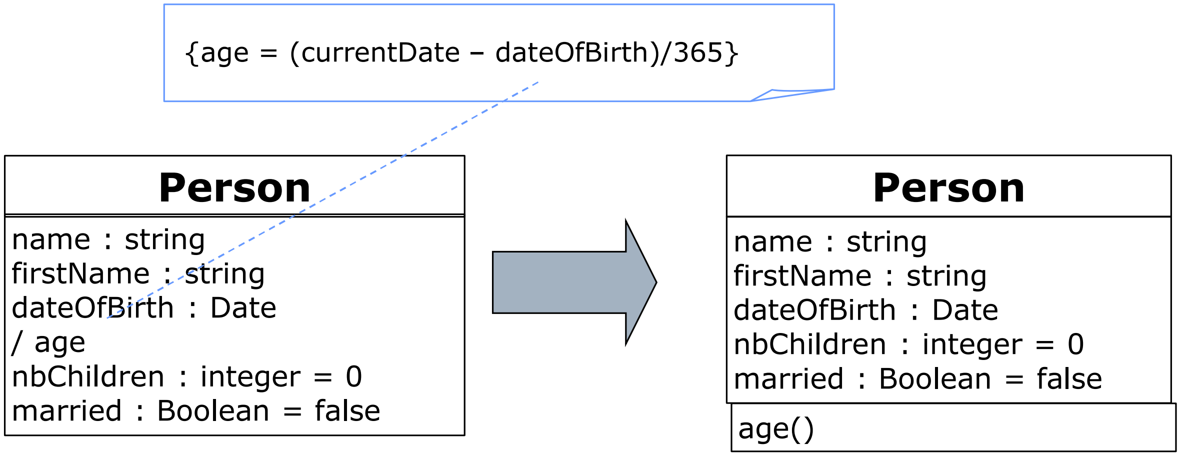
center : Point
radius : double

getCenter() : Point
setCenter(p : Point)
getRadius() : double
setRadius(r : double)
area() : real
move(p : Point)



Derived attributes

- Attributes can be deducted from other attributes
  - age of a person can be derived from date of birth



{age = (currentDate – dateOfBirth)/365}

Person

name : string
firstName : string
dateOfBirth : Date
/ age
nbChildren : integer = 0
married : Boolean = false

Person

name : string
firstName : string
dateOfBirth : Date
nbChildren : integer = 0
married : Boolean = false

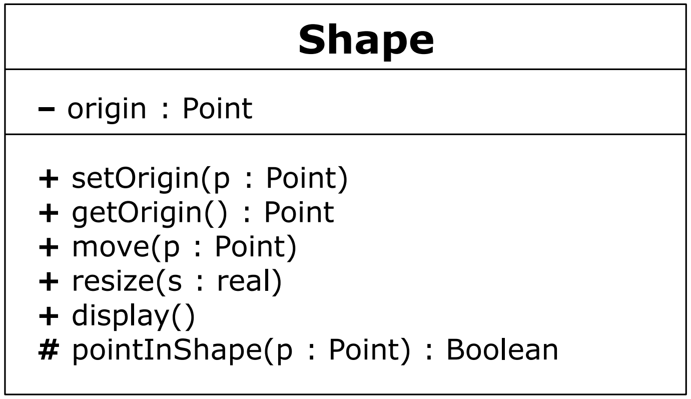
age()

High level design

Detailed design

Visibility

- Attributes and operations have the visibility
  - Public
    - visible outside the class
    - notation “ + ”
  - Protected
    - visible only to objects of the same class and objects of sub-classes
    - notation “ # “
  - Private
    - visible only to objects of the class
    - notation “ - “



Shape

− origin : Point

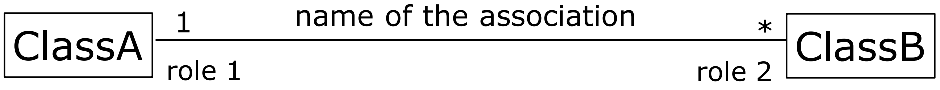
+ setOrigin(p : Point)
+ getOrigin() : Point
+ move(p : Point)
+ resize(s : real)
+ display()
# pointInShape(p : Point) : Boolean

Relationship types

- Relationships between classes
  - Association
    - Semantic relation between classes
  - Inheritance
    - A class can inherit one or more classes
  - Aggregation
    - An association shows a class is a part of another class
  - Composition
    - A strong form of aggregation
  - Dependency
    - shows the dependency between classes

Association

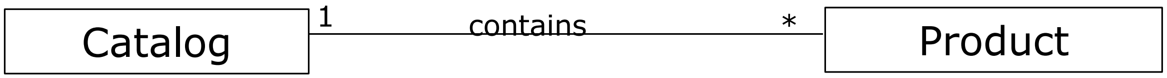
- An association
  - is used to show how two classes are linked together
  - expresses a bidirectional semantic connection between classes
  - is an abstraction of the links between instances of classes
  - Notation



  - Each end of an association is called a **role**
    - A role shows the purpose of the association
    - A role can have
      - an name
      - an expression of **multiplicity**

Association

- Multiplicity
  - defines how many instances of a class A are associated with an instance of class B



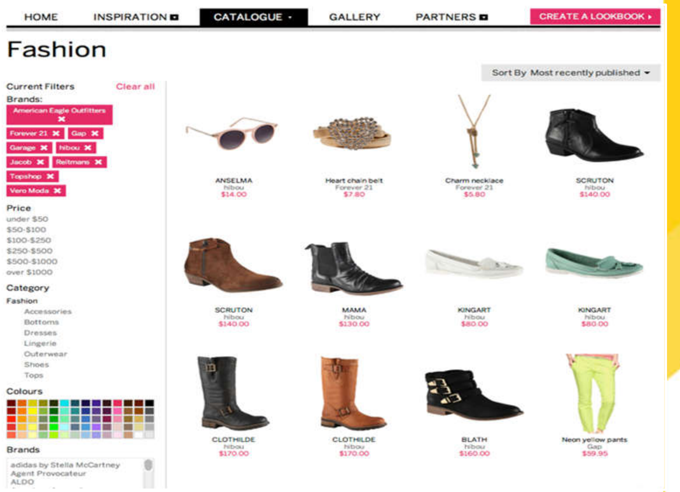
Catalog

1

contains

*

Product

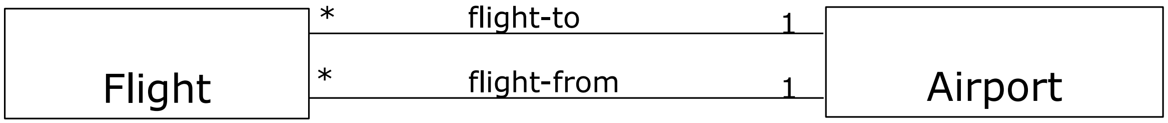


- Different expressions of multiplicity

1	:	one and only one
0..1	:	zero or only one
m..n	:	from m to n (integer, n >= m >= 0)
n	:	exactly n (integer, n >= 0)
*	:	zero or many
1..*	:	from one to many

Association

- Multiple associations
  - Two classes can have several associations between them



Flight

* flight-to 1

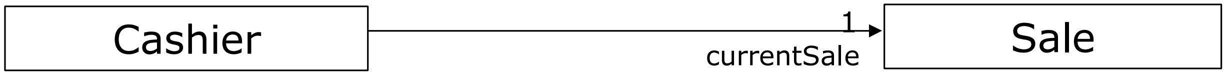
* flight-from 1

Airport

Dien Bien
HA NOI
Hai Phong
Vinh
Dong Hoi
Hue
Da Nang
Chu Lai
Pleiku
Quy Nhon
Tuy Hoa
Buon Ma Thuot
Da Lat
Nha Trang
HO CHI MINH city
Rach Gia
Phu Quoc
Can Tho
Con Dao
Ca Mau

Association

- Directional association and attributes
  - By default, the associations are bi-directional
  - However, associations can be directional
    - Example



Cashier

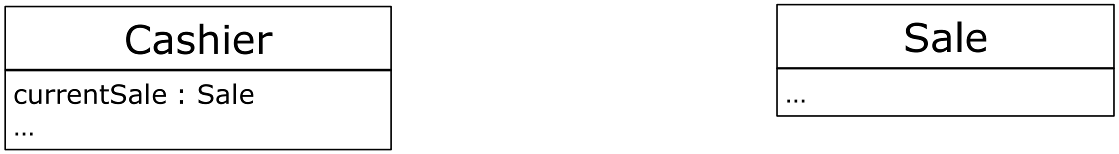
1

currentSale

Sale

- The navigability pointing from *Cashier* to *Sale* shows that an attribute with *Sale* type
- This attribute is called *currentSale*

- Another form of representation: use of attributes



Cashier

currentSale : Sale
…

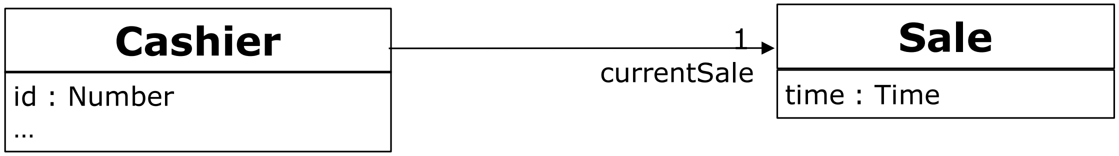
Sale

…

Association

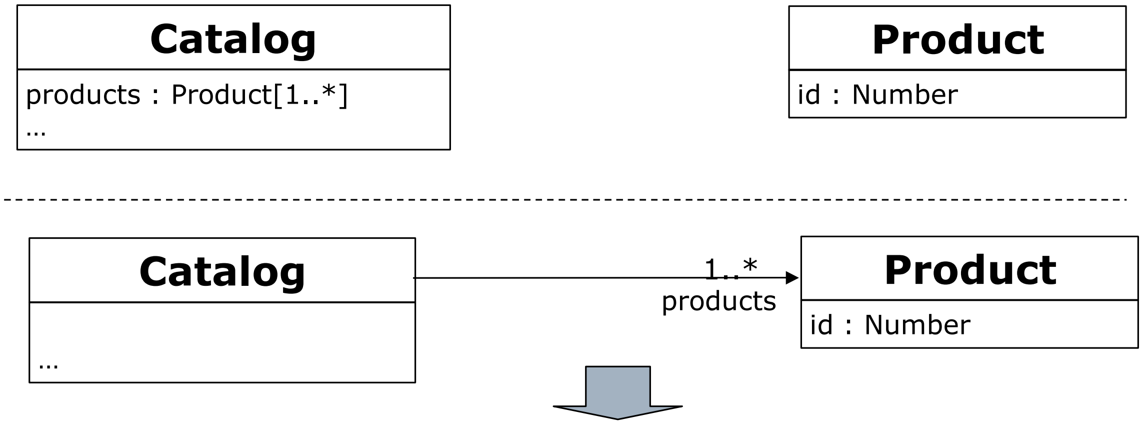
- Directional association and attributes
  - When do we use the directional association or attribute?
    - We use the attribute for “primitive” data types, such as Boolean, Time, Real, Integer, …
    - We use the directional association for other classes
      - To better see the connections between classes
    - It is just to better represent, these two ways are semantically equivalent

    - Example

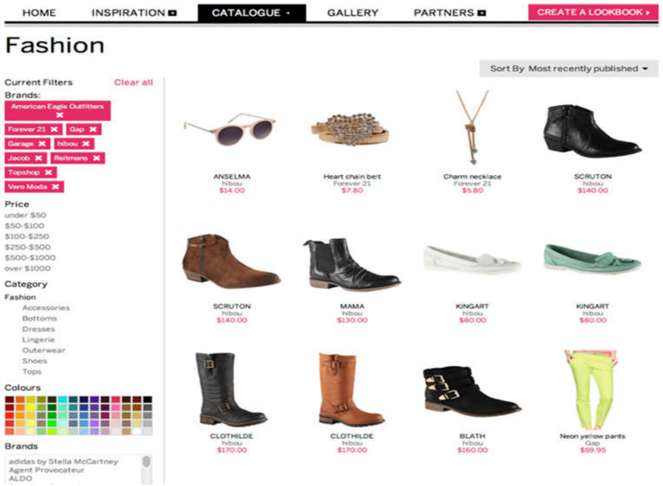


Association

- Directional association and attributes
  - Another example

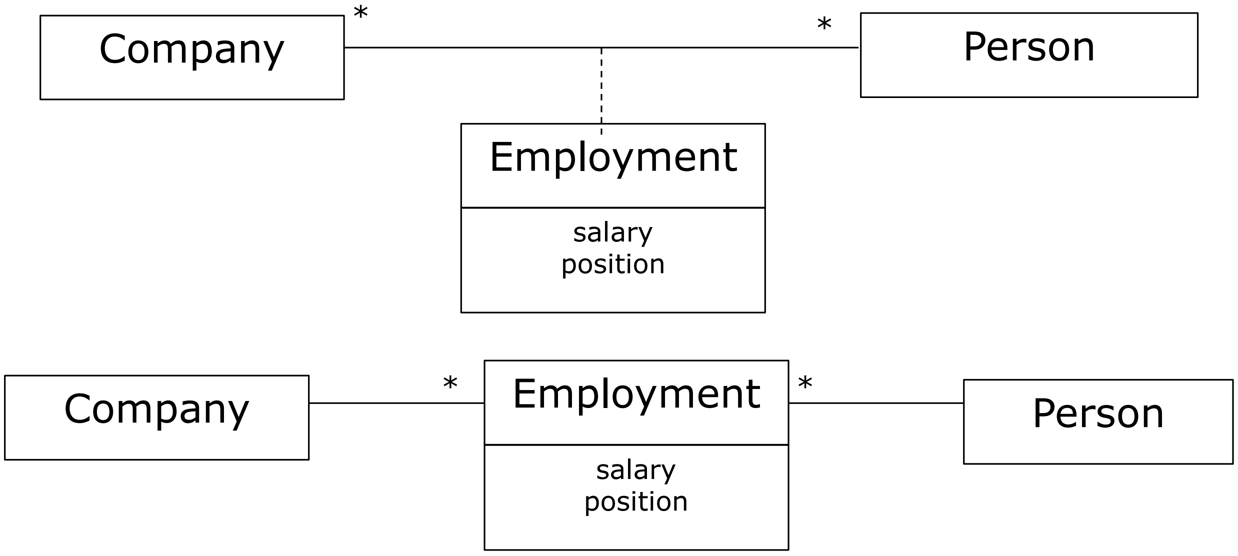


```
public class Catalog {
        private List<Product> products =
                        new ArrayList<Product>();
// ...
}
```



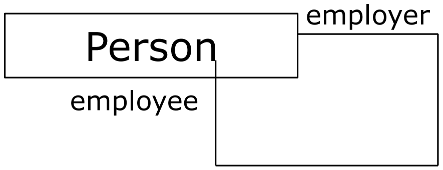
Association

- Association classes
  - An association class allows an association to be considered as a class
    - When an attribute cannot be attached to any of the two classes of an association
  - Example



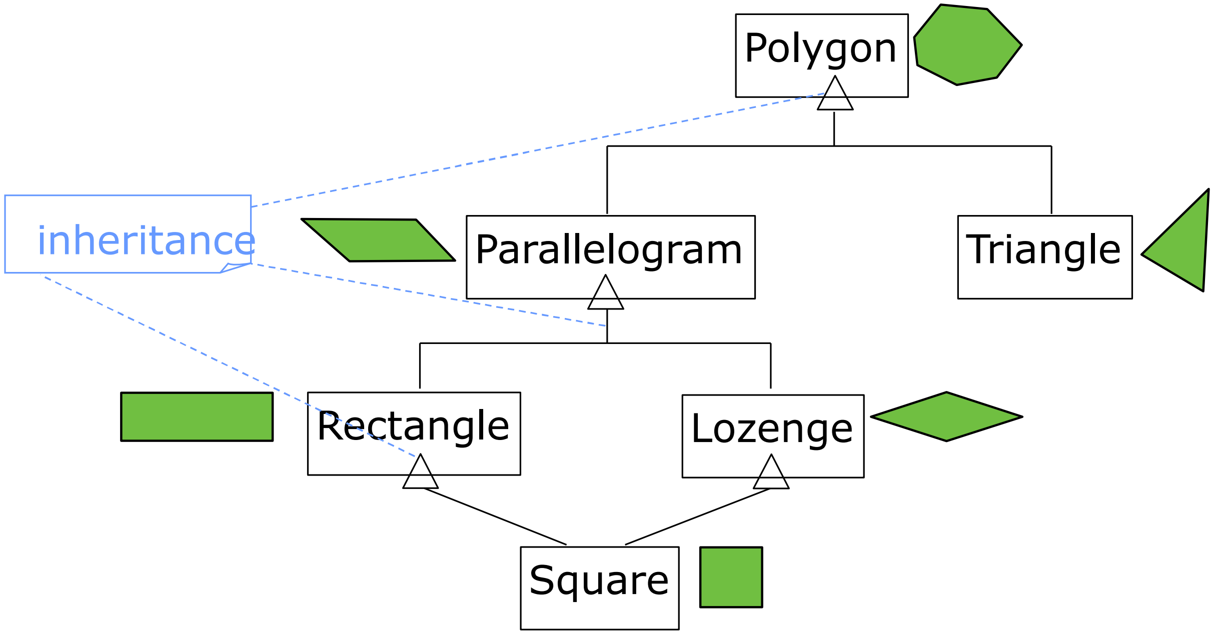
Association

- A class can be associated to itself
  - example



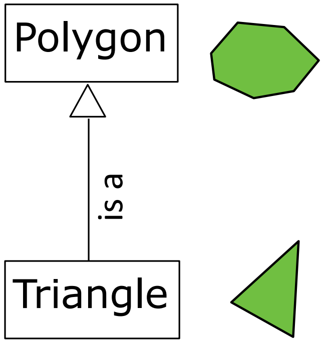
Inheritance

- A class can have several sub-classes



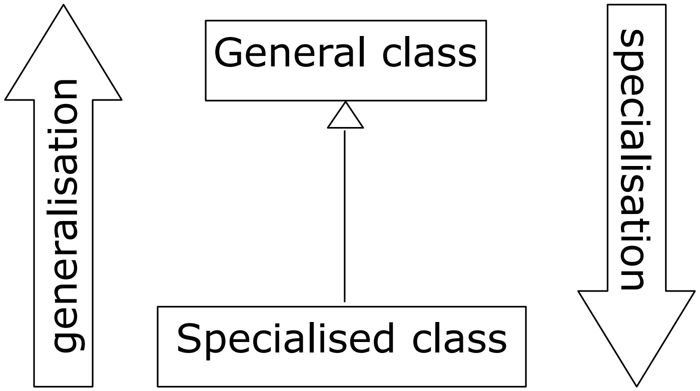
Inheritance

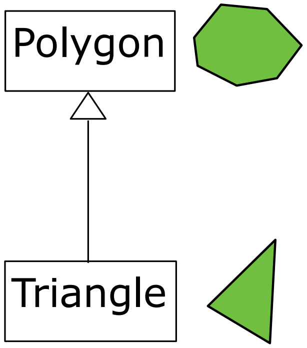
- Substitution principle
  - All subclass objects can play the role of an object of its parent-class
  - An object of a subclass can override an object of its superclass
- Informally
  - A subclass is a kind of superclass
- Example
  - A triangle is a polygon



Inheritance

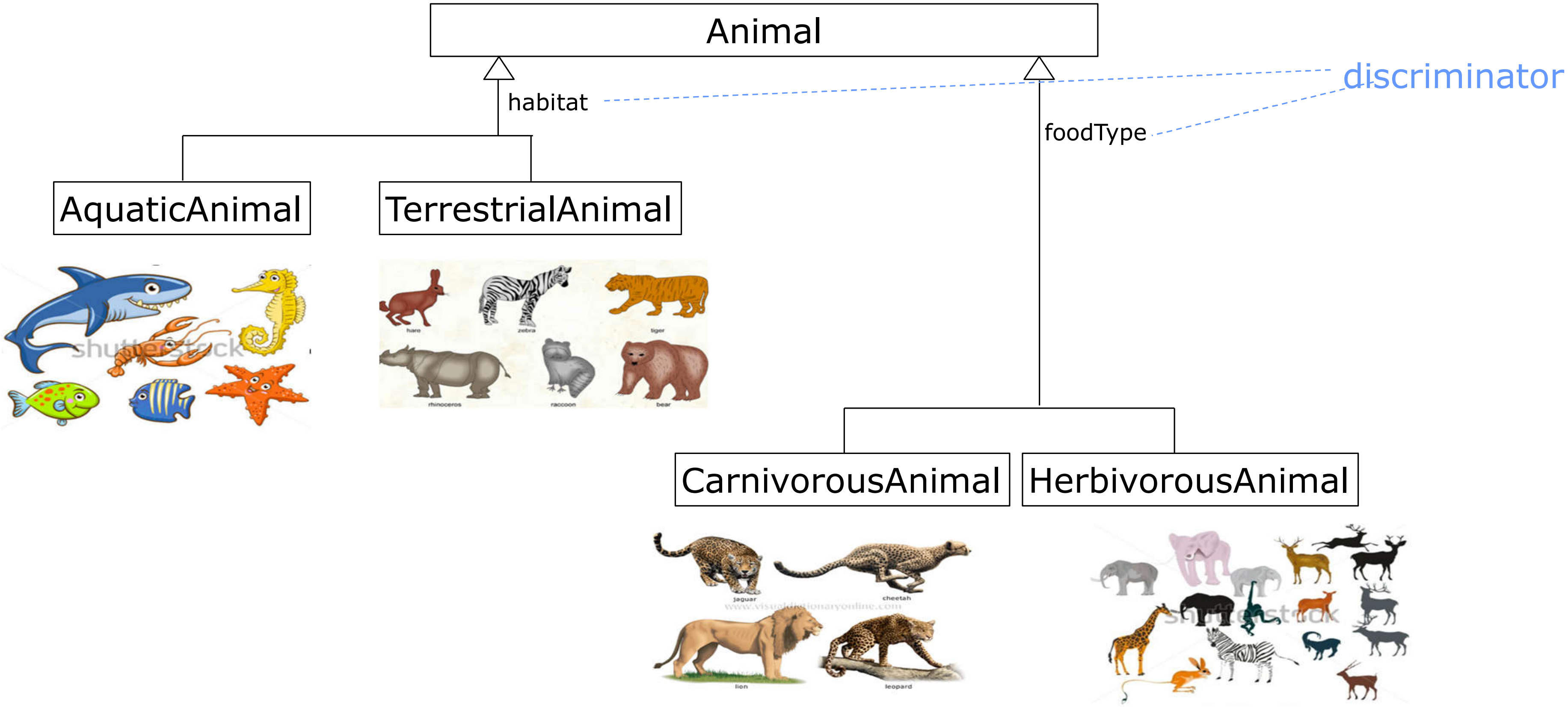
- The subclasses are also called **specialised classes**
- Parent-classes are also called **general classes**
- The inheritance is also called the **specialisation** or **generalisation**





Inheritance

- The (optional) **discriminator** is a label describing the criterion that the specialisation bases on



Animal

habitat

foodType

discriminator

AquaticAnimal

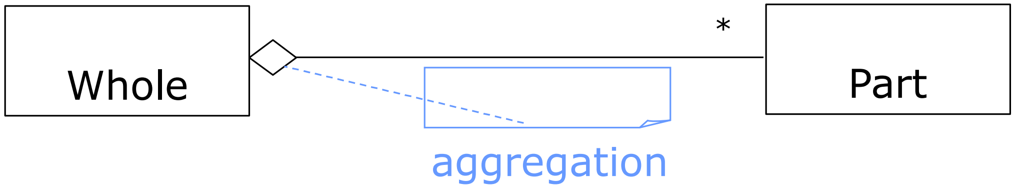
TerrestrialAnimal

CarnivorousAnimal

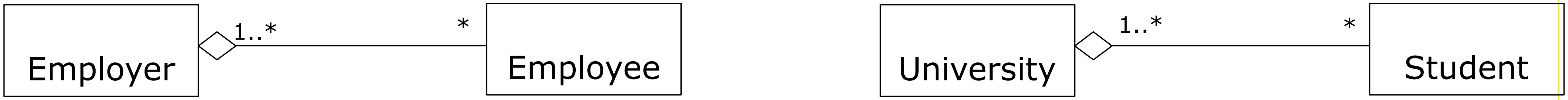
HerbivorousAnimal

Aggregation

- An aggregation is a form of association that expresses a stronger (than normal association) coupling between class
- An aggregation is used between two classes
  - master and slave: “belongs to”
  - whole and part: “is a part of”
- Notation
  - The symbol denoting the place of aggregation of the aggregate side

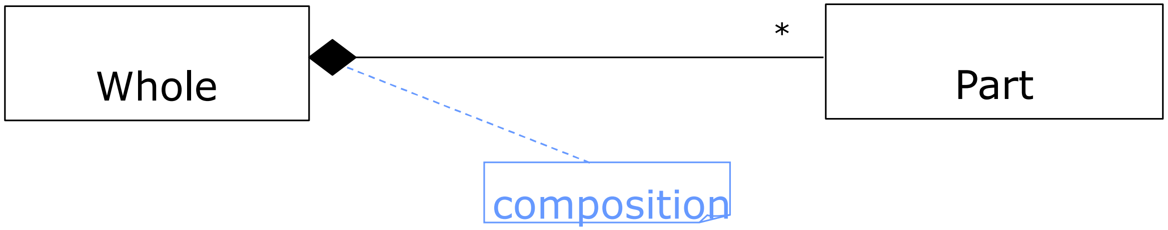


- Examples

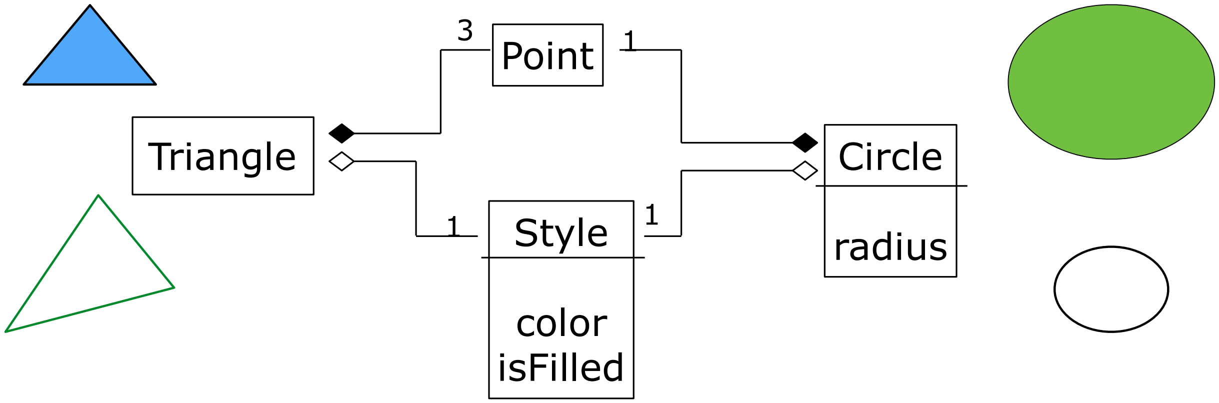


Composition

- A composition is a strong form of aggregation
- A composition is also a “whole-part” relationship but the aggregate is stronger
  - If the whole is destroyed then parts will be also destroyed

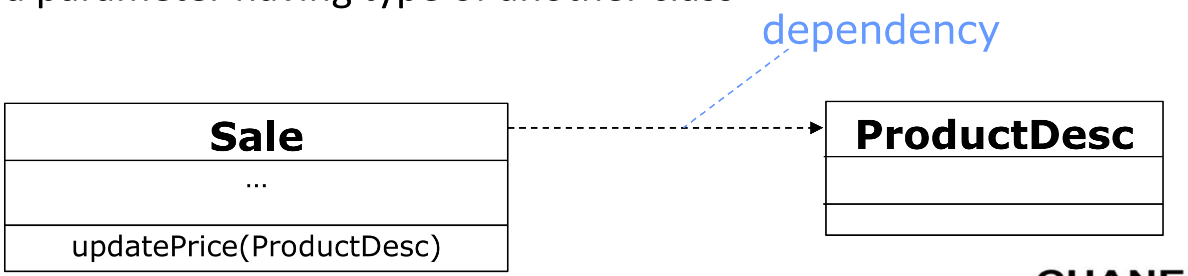


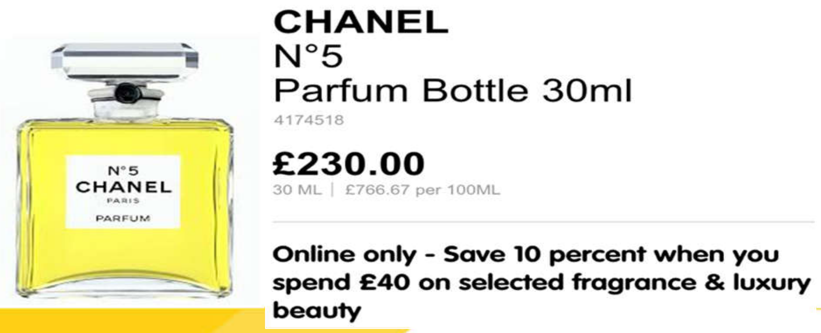
- Example



Dependency

- A class may depend on another class
- The dependency between classes can be implemented in different ways
  - Having an attribute with the type of another class
  - Sending a message using an attribute, a local variable, a global variable of another class or static methods
  - Receiving a parameter having type of another class
- Example





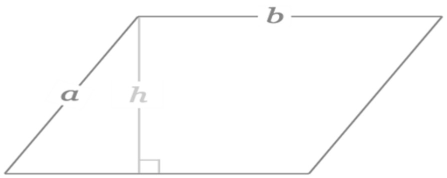
CHANEL
N°5
Parfum Bottle 30ml
4174518

£230.00
30 ML | £766.67 per 100ML

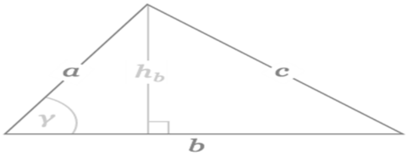
Online only - Save 10 percent when you spend £40 on selected fragrance & luxury beauty

Abstract class

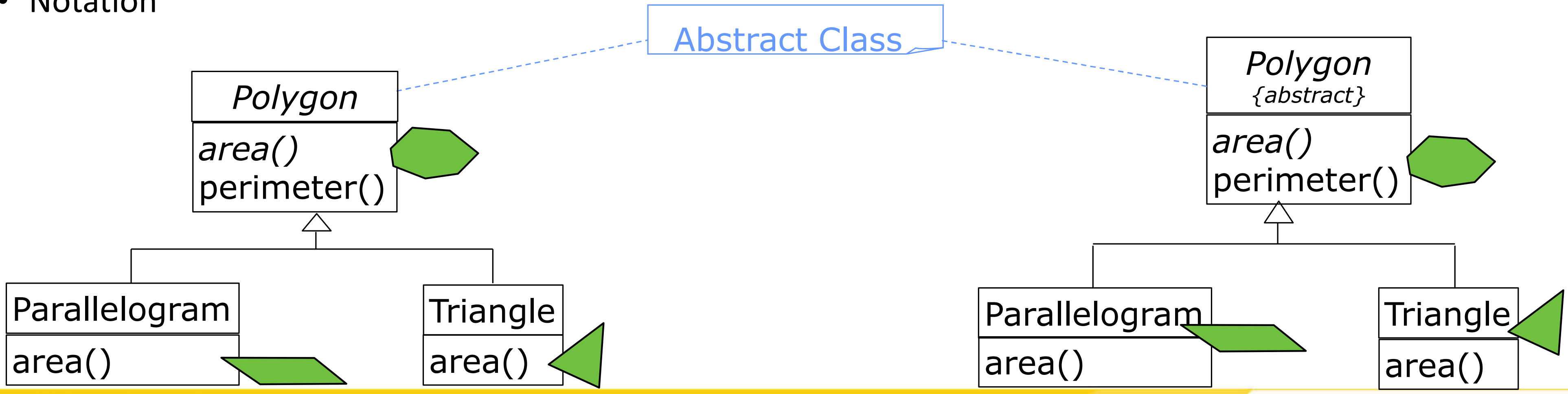
- An abstract class is a class that has no instances/objects
  - inheritance: *area()*, perimeter()
  - polymorphism: *area()*
    - Parallelogram = b * h



Triangle = (h * b) / 2



- Notation



Abstract Class

Polygon

area()
perimeter()

Parallelogram

area()

Triangle

area()

Polygon
{abstract}

area()
perimeter()

Parallelogram

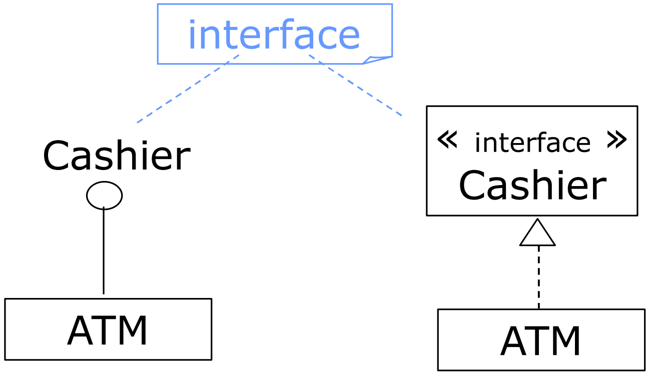
area()

Triangle

area()

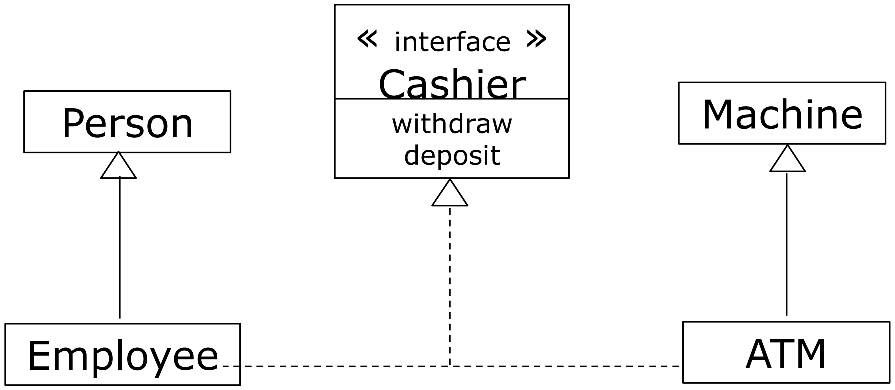
Interface

- An interface
  - describes a portion of the visible behaviour of a set of objects
  - is very similar to an abstract class that contains only abstract operations
  - **specifies only the operations without implementation**
- Two notations



Interface

- Example



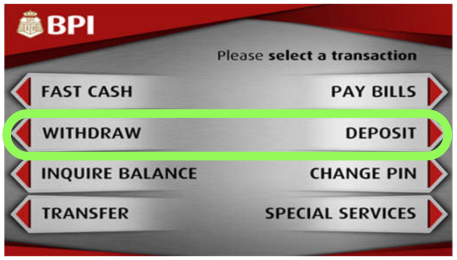
« interface »
Cashier
withdraw
deposit

Person

Machine

Employee

ATM



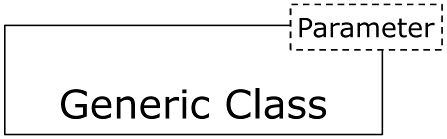
1 Insert your ATM card
Select "Other Services" option
Follow simple on-screen instructions

2 Insert cash
Follow simple on-screen instructions

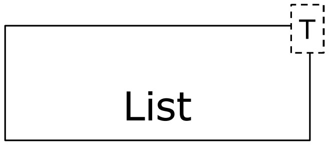
3 Amount is already on your account!

Generic class

- A generic class (or parameterised) allows to consider the types of data as parameters
- Generic classes are often used for the types of collection classes: vector, table, stack, …
- Notation



- Example



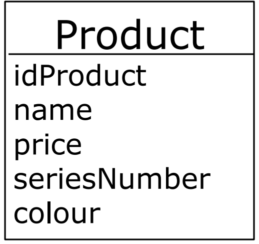
- “template” in C++
- Generic type in Java
  - List<Integer> **intList** = new ArrayList<Integer>();

→ intList =

5	3	9	11

Description class

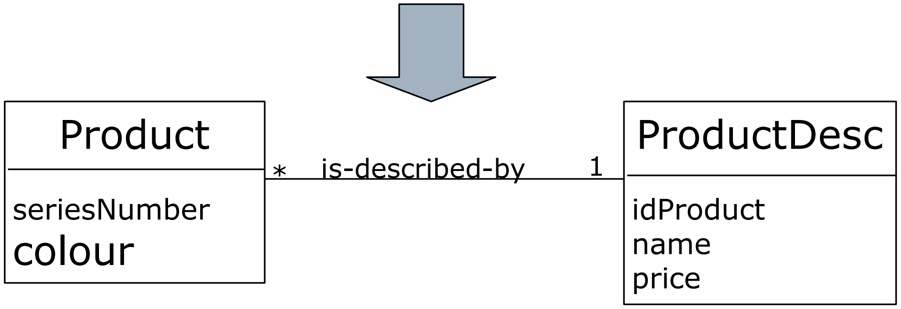
- Description class contains information describing the objects
  - increase cohesion, reusability
- Example
  - Description of a product with certain information



Product

idProduct
name
price
seriesNumber
colour

Solution 1 : no description class



Product

seriesNumber
colour

* is-described-by 1

ProductDesc

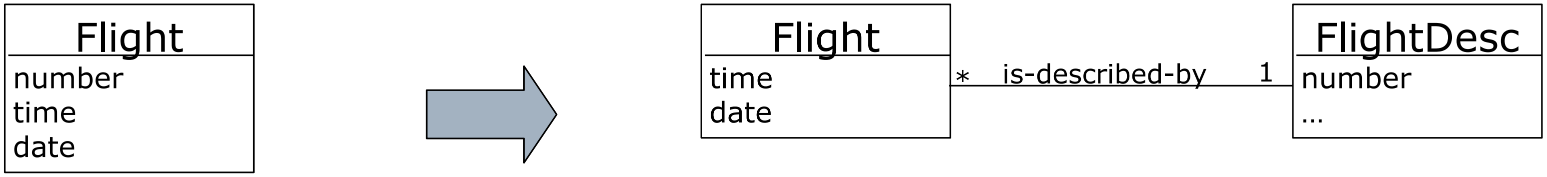
idProduct
name
price

Solution 2 : use description class

Description class

- When do we use description class?
  - Reduce redundant information
  - Avoid repetition of information
  - Describe some objects independent of real-world objects
  - Keep the information in the object even if real-world objets are removed

- Example
  - Describe the information (number, time, date, …) of a flight



Flight
number
time
date

Flight
time
date

* is-described-by 1

FlightDesc
number
…

Description class ? Description class ? Description class ?

VDA6

Building class diagrams

- Class diagrams are built at different stages and at different levels
  - Domain model
    - Model a set of **conceptual classes (concepts)** and the relationships between them
  - System model
    - Model all classes and their relationships, including the architecture classes and user interface classes

VDA5

Slide 39

VDA5 stages? levels?

example?

example?

VO Duc An, 04/06/2015

VDA6 Follow this example:
http://www.simventions.com/whitepapers/uml/3000_borcon_uml.html

http://www.powershow.com/view1/1c39d0-ZDc1Z/2120_Fitting_the_UML_into_Your_Development_Process_powe

VO Duc An, 18/06/2015

Building class diagrams

- Suggested steps to build class diagrams
  - Build the domain model
    - Identify conceptual classes
    - Identify the relationships
    - Identifier the attributes
    - Identify the inheritances and the interfaces
    - Determine the main responsibilities of each conceptual class
  - Build the system model
    - Introduce new classes
    - Detail the attributes
    - Detail the relationships
    - Decide the operations of each class
  - During the development, refine progressively the class diagrams until satisfaction
    - Add or remove classes, attributes, operations, relationships

Building class diagrams

- Building domain model
  - The domain model is the important model in object-oriented analysis
  - This model is also called the analysis class diagrams

VDA8

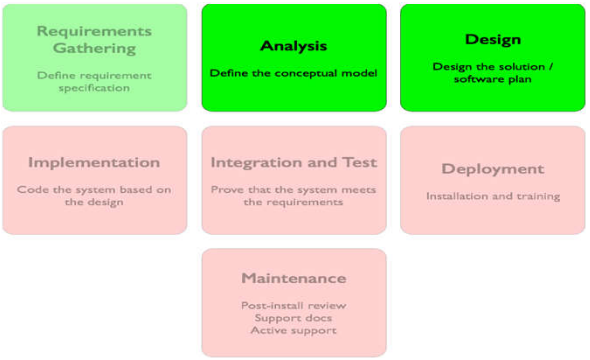
Slide 41

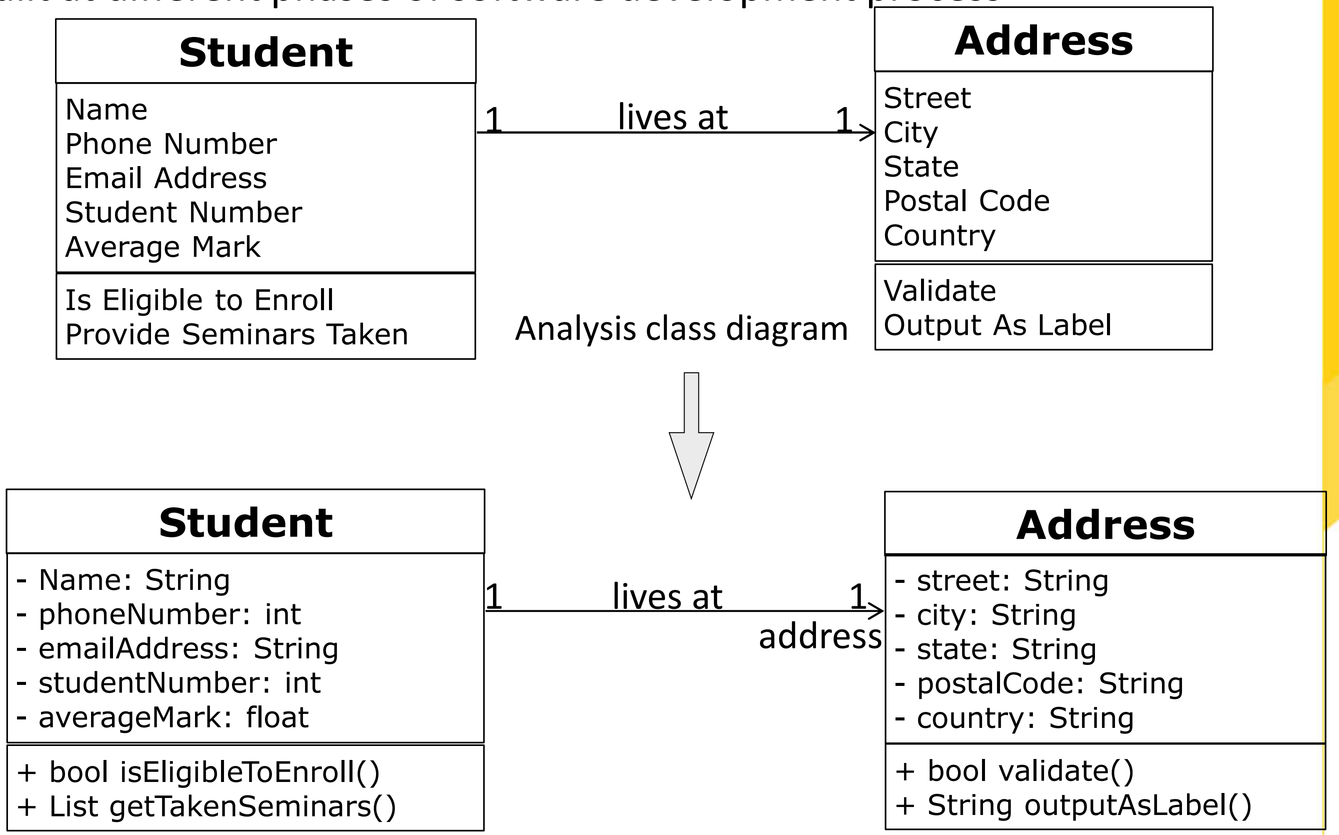
VDA8 example?

VO Duc An, 15/06/2015

Building class diagrams

- Class diagrams are progressively built at different phases of software development process


Requirements Gathering
Define requirement specification
Analysis
Define the conceptual model
Design
Design the solution / software plan
Implementation
Code the system based on the design
Integration and Test
Prove that the system meets the requirements
Deployment
Installation and training
Maintenance
Post-install review
Support docs
Active support


Student
Name
Phone Number
Email Address
Student Number
Average Mark

Is Eligible to Enroll
Provide Seminars Taken

Address
Street
City
State
Postal Code
Country

Validate
Output As Label

Analysis class diagram

Student
- Name: String
- phoneNumber: int
- emailAddress: String
- studentNumber: int
- averageMark: float

+ bool isEligibleToEnroll()
+ List getTakenSeminars()

Address
- street: String
- city: String
- state: String
- postalCode: String
- country: String

+ bool validate()
+ String outputAsLabel()

Design class diagram (for Java implementation)

Building class diagrams

- Identifying classes
  - The question “How to find classes?”
  - The concepts in the studied domain can be also classes
    - These concepts are called **conceptual classes**
    - So, we **firstly** identify the conceptual classes, and **then other** classes are added during the development

- The principles for finding conceptual classes
  - Use of a **list of categories**
  - Identification of **nouns**

Building class diagrams

- Identifying classes
  - Use of a list of categories

Categories of conceptual classes	Examples
transaction (of business)	Reservation, Payment
product ou service relating to the transaction	Product, Flight
where transactions are recorded?	Cash desk, Cash
actors of use-cases	Cashier, Customer
location (of service, of transaction)	Station, Store
important events	purchase
physical objects	Car
description of things	Description of products
catalog	Product catalog
containing things	Store
other collaboration systems	Bank, database
organisations	University
policy, principle	Tax
…	

Building class diagrams

- Identifying classes
  - Identification of **nouns**
    - Review written documents such as specification or description of use-cases
    - Extract names and consider them as conceptual class candidates
    - Remove the nouns which
      - are redundant
      - are vague or too general
      - aren’t conceptual classes by experience and knowledge in the context of the application

Building class diagrams

- Identifying classes
  - Identification of **nouns** from use-case spec
    - Example

Actions of actor	**Actions of system**
• The **customer** comes to the **cash desk** with the **products** to buy	
• The **cashier** encodes **the identifier** of each **product** If a **product** has more than one **item**, the **cashier** inputs the number of **items**	• The **cash desk** displays the **description** and price of the **product** This **number** is displayed
• After having encoded all of the **products**, the **cashier** signals the end of the **purchase**	• The **cash desk** calculates and displays the total amount that the **customer** has to pay
• The **cashier** announces the total amount to the customer	
• The **customer** pays	• The **cash desk** displays the **balance**
• The **cashier** input the amount of **money** paid by the **customer**	

Building class diagrams

- Identifying classes
  - Identification of **nouns**
    - Example (continue)

Actions of actor	Actions of system
• The **cashier** receives the cash **payment**	• The **cash desk** prints the **receipt**
• The **cashier** gives **change** to the **customer** and the **receipt**	• The **cash desk** saves the **purchase**
• The customer leaves the **cash desk** with the bought **products**	

Building class diagram

- Candidate classes from nouns identified from use-case description
  - **customer, cash desk, product, item, cashier, purchase, change**

Building class diagrams

- Identifying the **relationships and attributes**
  - Starting with **central classes** of the system
  - Determining the attributes of each class and associations with other classes
  - Avoiding adding too many attributes or associations to a class
    - To better manage a class

Building class diagrams

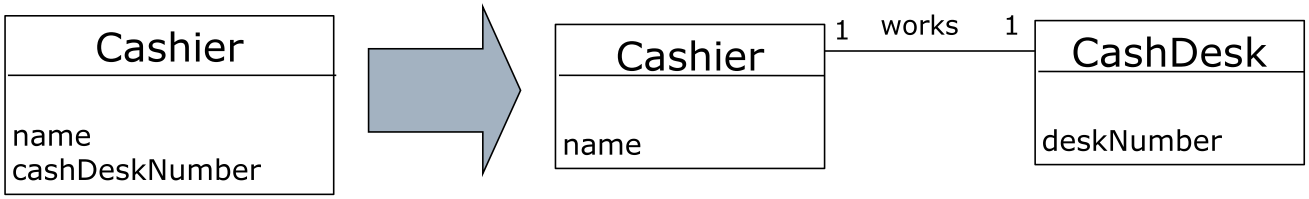
- Identify the relationships
  - A **association** should exist between class A and class B, if
    - A is a service or product of B
    - A is a part of B
    - A is a description for B
    - A is a member of B
    - A is connected to B
    - A possesses B
    - A controls B
    - …
  - Specify the **multiplicity** at each end of the association
  - Label associations

Building class diagrams

- Identifying attributes
  - For each class, determine the information needed to store according to the requirement specification or use-case
    - Example: Cashier needs an identifier, a name, …
  - Principle to determine attributes
    - **An attribute represents only data related to the class that owns the attribute**
    - If a subset of the attributes form a coherent group, it is possible that a new class is introduced
  - Determine only the names of attributes at this stage (i.e., analysis phase)

Building class diagrams

- Identifying attributes
  - Example
    - An attribute represents only data related to the class that owns the attribute



Cashier

name
cashDeskNumber

Cashier

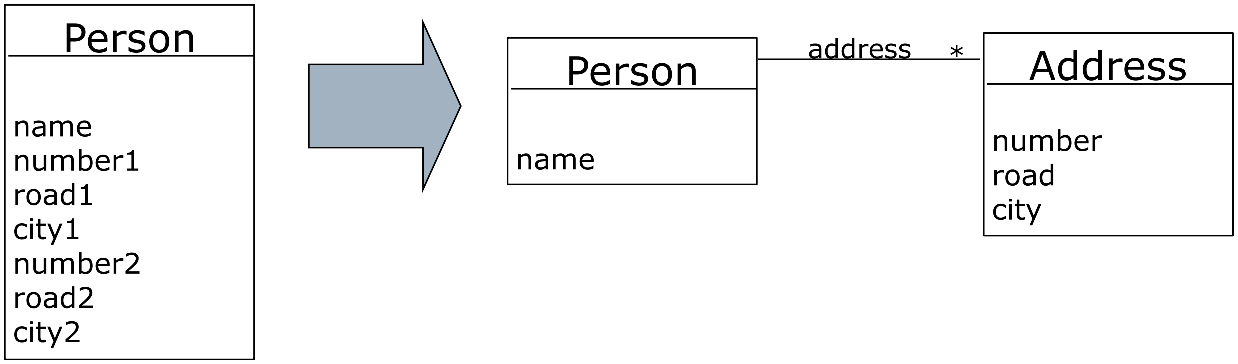
name

1 works 1

CashDesk

deskNumber

  - If a subset of the attributes form a coherent group, it is possible that a new class is introduced



Person

name
number1
road1
city1
number2
road2
city2

Person

name

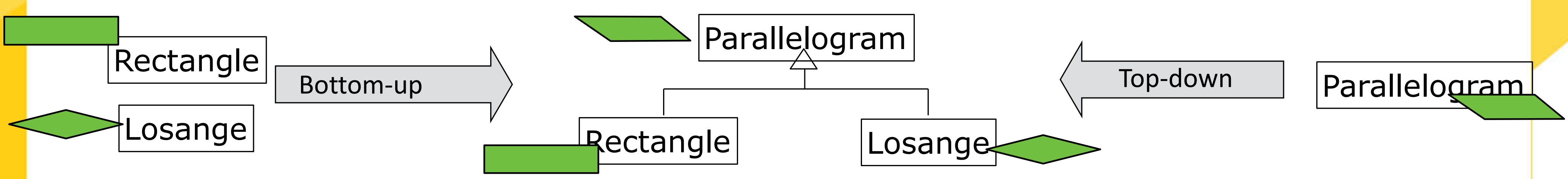
address *

Address

number
road
city

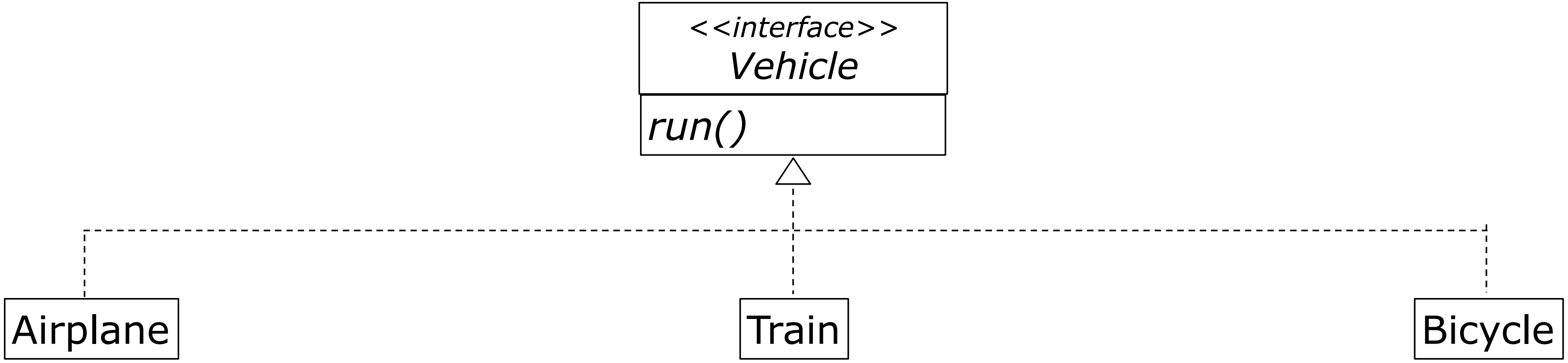
Building class diagrams

- Identifying **inheritances**
    - Two approaches
        - Bottom-up
            - Generalization: group similar classes to create super-classes
        - Top-down
            - Specialization: build sub-classes from existing general classes



Building class diagrams

- Identifying interfaces
  - Create interfaces rather than super-class, if
    - It is necessary to realise different implementations of the same class
    - Two classes to generate share the operations that are not similar
    - The class to generalise already has its own super-class



<<interface>>
Vehicle

run()

Airplane

Train

Bicycle

Building class diagram

- Determining the responsibilities of classes
  - A responsibility is one or several tasks that the system has to perform
  - **Each functional requirement must be attributed to one of the classes**
    - All the responsibilities must be attributed to one of the classes
    - If a class has too many responsibilities, it must be divided into several classes
    - If a class has no responsibility, it should be probably be useless
    - If responsibility cannot be assigned to any class, a new class can be introduced
  - The responsibilities can be determined by analysing the actions/verbs in the use-case specification.

Building class diagrams

- Developing **design class diagrams**
  - Basing on analysis class diagrams (domain models)
  - Detailing analysis class diagrams
    - Introducing new classes, if necessary
      - For example, an association of class becomes a new class
    - Detailing attributes
    - Adding and detail relationships
    - Determining operations/functions

Building class diagrams

- Detailing attributes
  - Determining the types of attributes
    - Using primitive types: boolean, int, real, …
    - Defining new type for an attribute (new class), if
      - It consists of several sections
      - It has other attributes
      - It is associated with other operations
  - Determining initial values if necessary
  - Determining the visibility of attributes

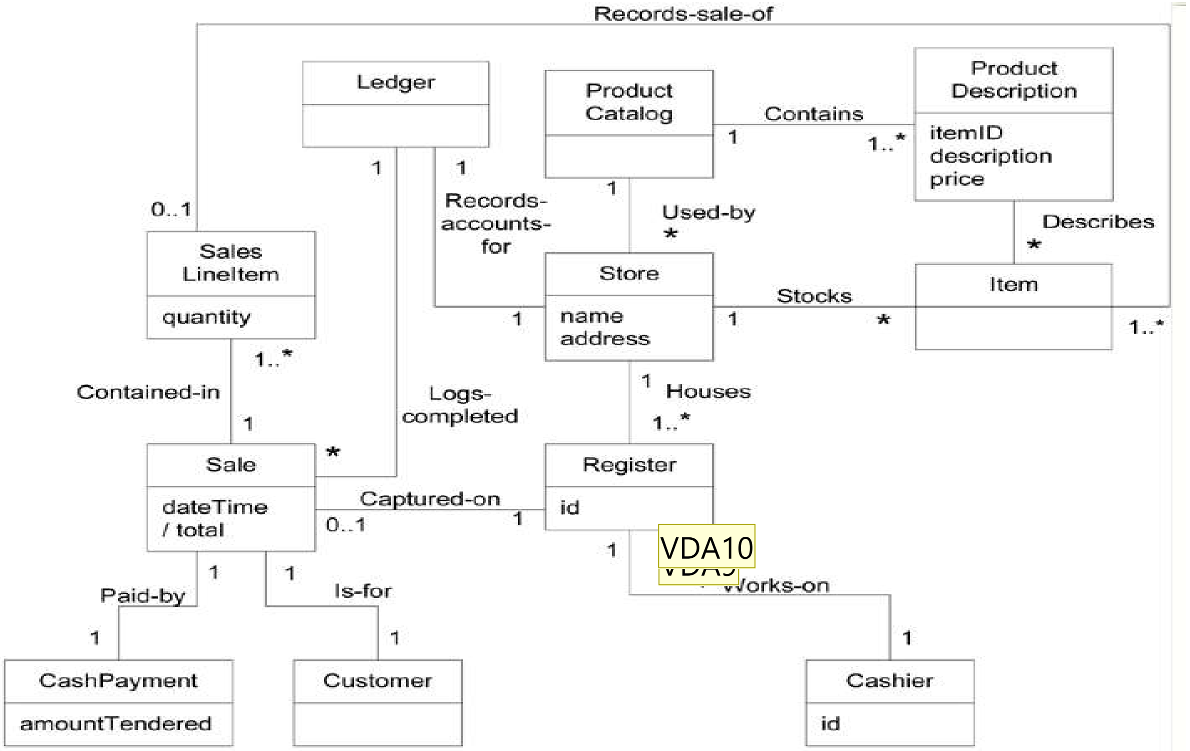
- Detailing relationships
  - Introducing relationships according to newly added classes
  - Specifying if an association is an aggregation or a composition
  - Naming the relationship
  - Giving the direction

Building class diagrams

- Determining the operations/functions of each class
  - getters and setters
  - Operations are used to achieve the identified responsibilities
  - A responsibility can be carried out by several operations
  - Determining the visibility of operations
    - Essential operations carrying out responsibilities are declared “public”
    - Operations serving only in the class are declared “private” or “protected” if the class should be inherited

Building class diagrams

- Example: supermarket cash register system



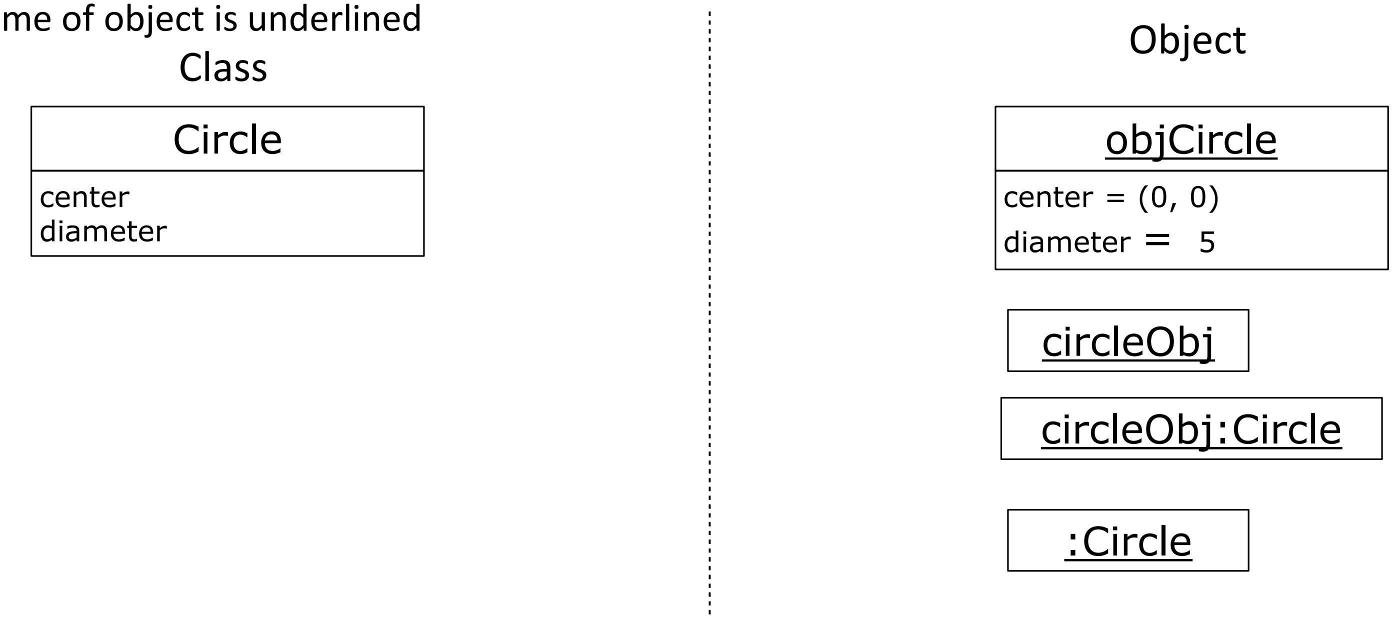
Slide 59

VDA9 review this title
VO Duc An, 09/06/2015

VDA10 an example to show step by step
concept diagram -> analysis class diagram -> design class diagram
VO Duc An, 16/06/2015

Object diagrams

- Objects
  - Objects are instances of classes
  - Notation
    - Values of attributes can be indicated
    - Name of object is underlined



Class

Circle

center
diameter

Object

objCircle

center = (0, 0)
diameter = 5

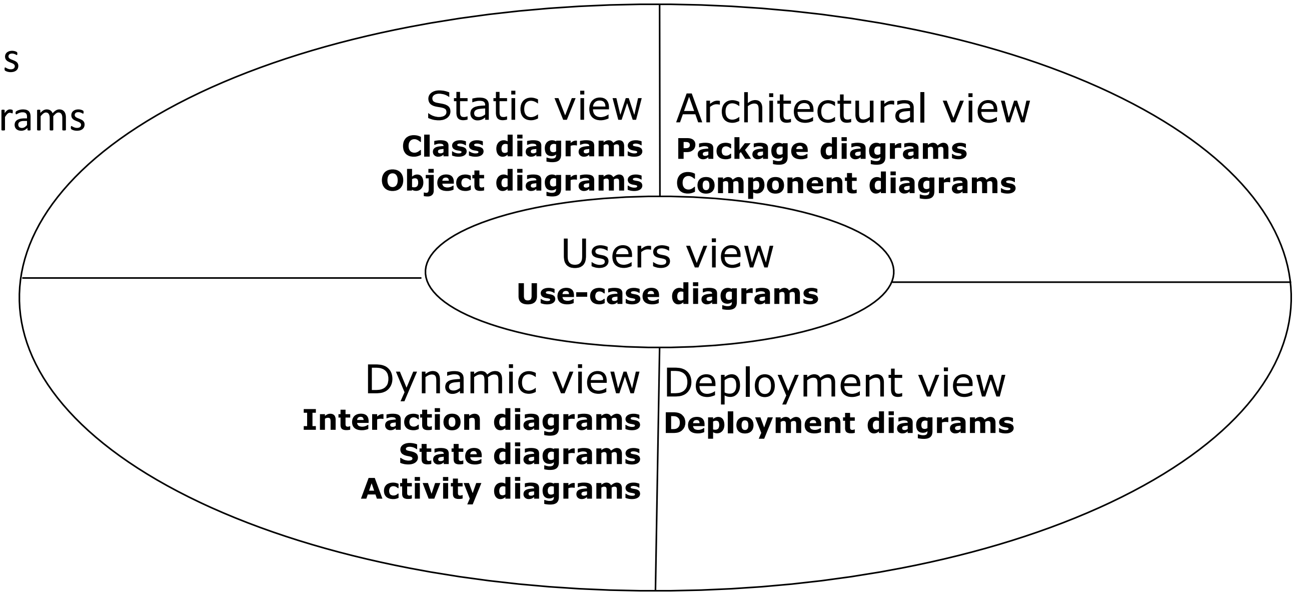
circleObj

circleObj:Circle

:Circle

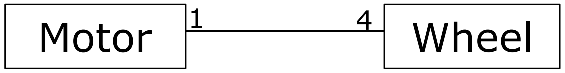
Object diagrams

- Objects
  - Three types of diagrams with objects
    - Static view
      - Object diagrams
    - Dynamic view
      - Sequence diagrams
      - Collaboration diagrams

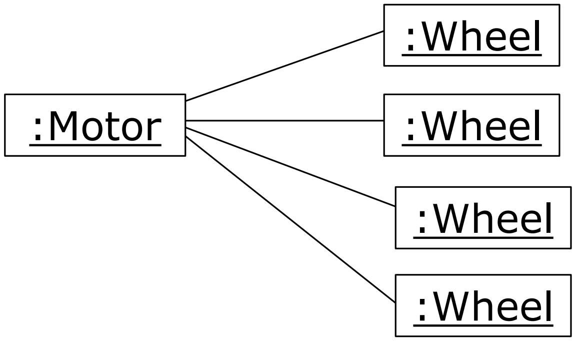


Object diagrams

- Object diagrams
  - represent a set of objects and links between them
  - are static views of instances of the elements appearing in class diagrams
- An object diagrams is an instance of a class diagram



Class diagram



Object diagram

Project (1)

- Divide groups of 4-5 students
- Each group chooses a problem
- Building Analysis class diagrams : Choose one of the following tools:
  - Microsoft Visio
  - StarUML: http://staruml.io/
  - Argo UML: https://argouml.jaleco.com/
  - Lucidchart: https://www.lucidchart.com/pages/examples/uml_diagram_tool
  - Or Others....

Project (2)

- Divide groups of 4-5 students
- Each group chooses a problem
- Building Design class diagrams : Choose one of the following tools:
  - Microsoft Visio
  - StarUML: http://staruml.io/
  - Argo UML
  - Lucidchart: https://www.lucidchart.com/pages/examples/uml_diagram_tool
  - Or Others...

Chapter 5.
Modelling Static Structure