Universitatea Tehnica din Cluj-Napoca
Departament Calculatoare

Programming Techniques in Java

Design View: UML Diagrams

T. Cioara, C. Pop, V. Chifu
2025

Models

What is a model?

View the design of the application

Simplification of reality

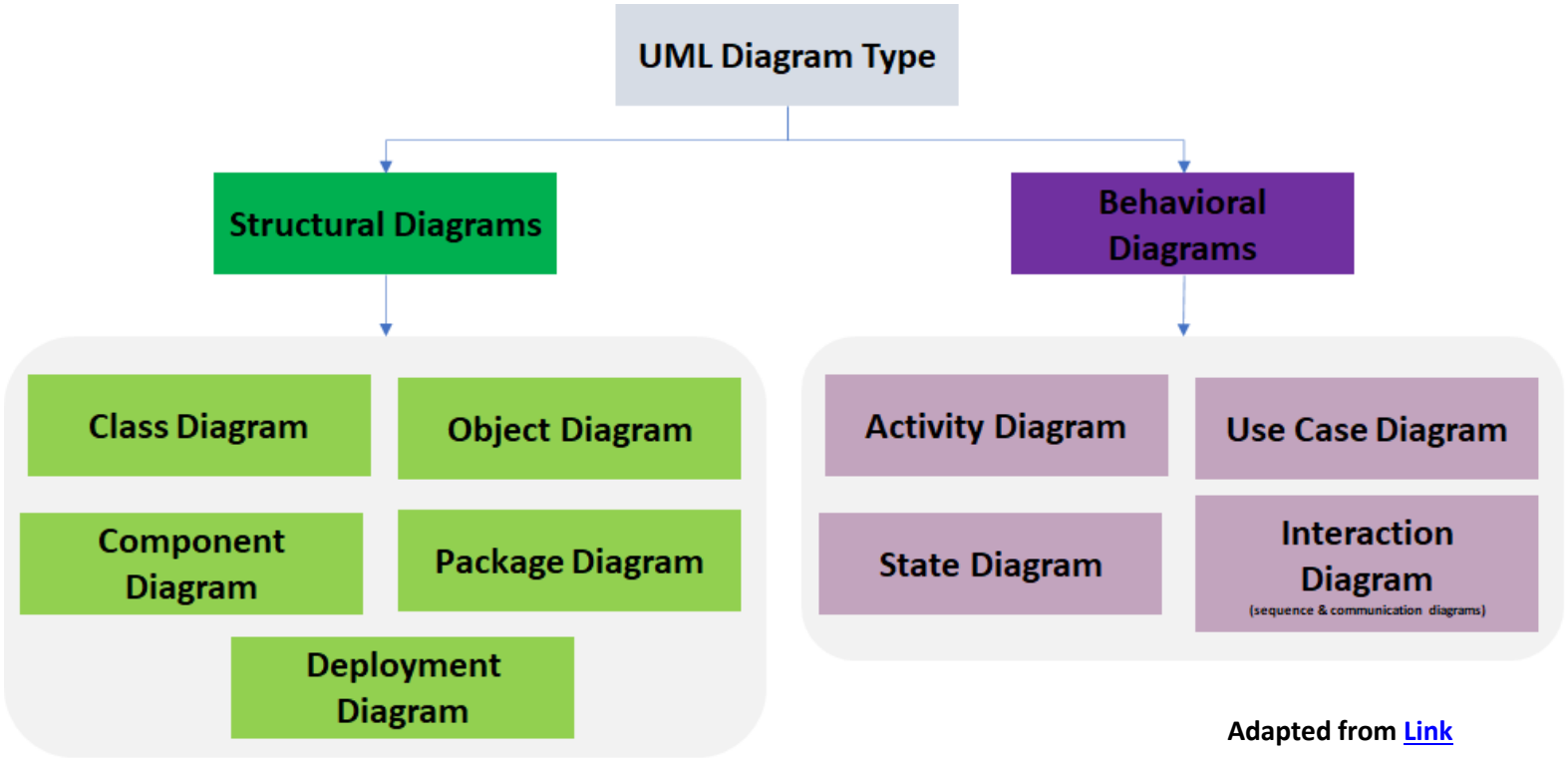
Start of coding

What are models for?

- Checking the requirements and domain knowledge
- Detail the design of a system
- Exploring multiple solutions (e.g., cost, time, etc)
- Guiding the development process
- Making changes and reducing dependencies

UML Diagrams

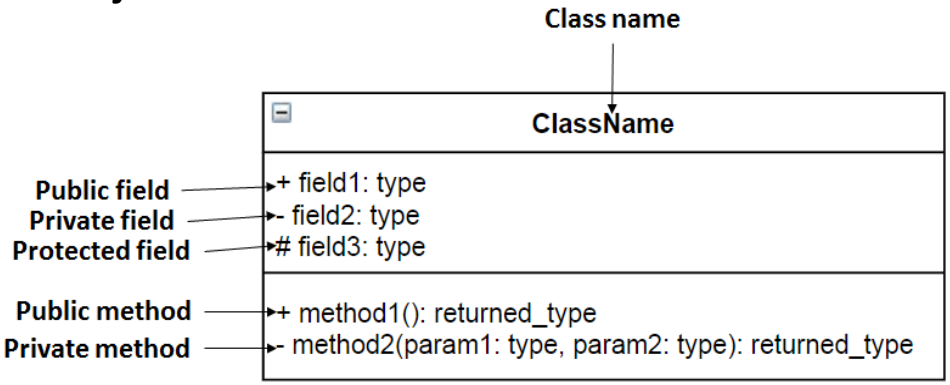
- Unified Modelling Language
  - visualizing, specifying, constructing, and documenting
  - artifacts of a software system



Adapted from Link

Class and Object Diagrams

- Illustrate the logical structure of the system
  - Classes (attributes, operations) and objects
  - Relationships
    - associations
    - aggregation/ composition
    - generalization (inheritance)
    - realization
    - Dependency



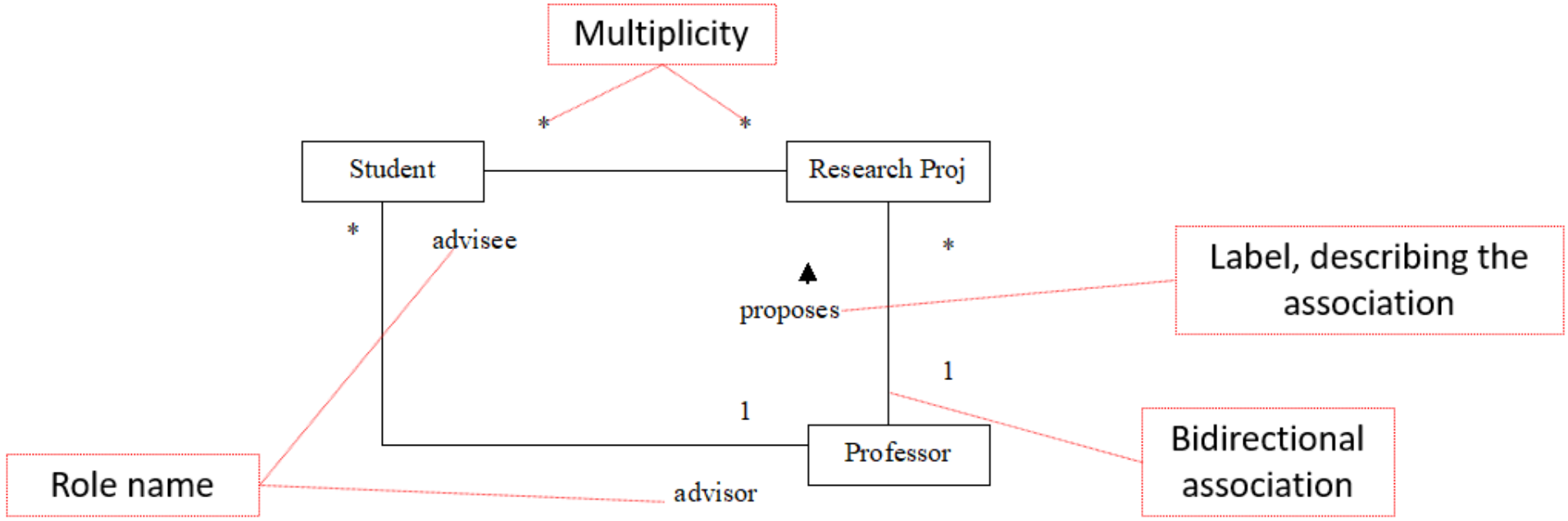
UML representation of a class

- Implementation perspective upon the system being developed

An object diagram represents a snapshot of the instances in a system and the relationships between the instances

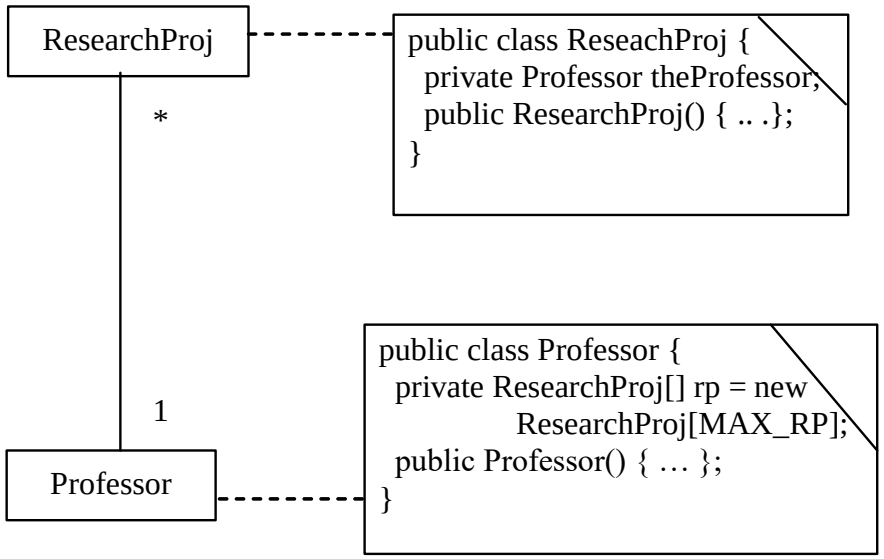
Class and Object Diagrams

- Associations
  - Binary relationships among classes

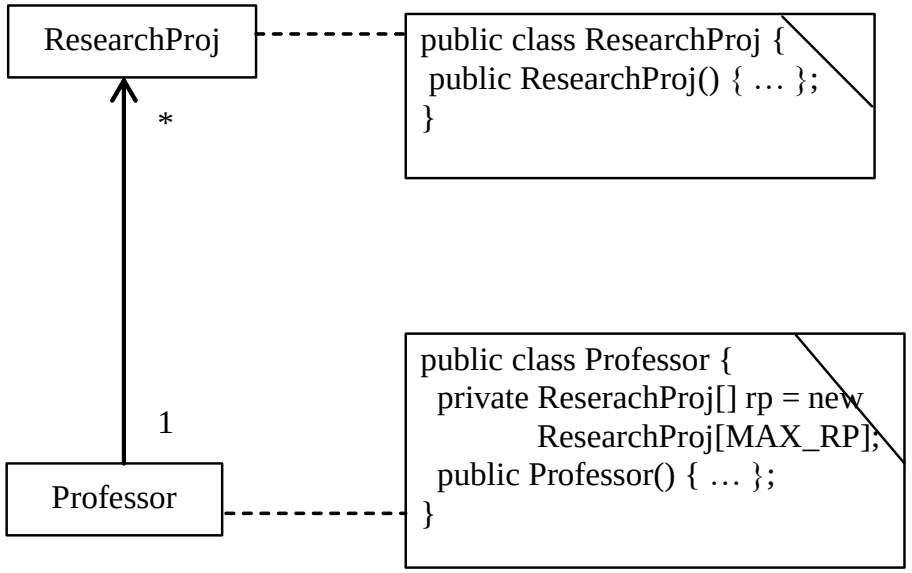


Class and Object Diagrams

- Associations
  - Bi-directional vs uni-directional



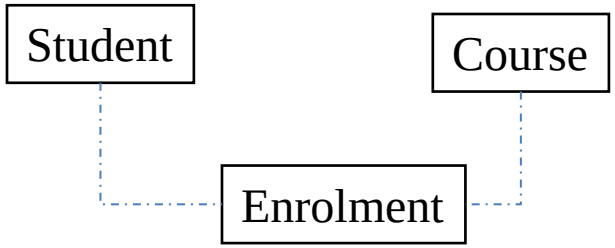
Java class skeleton



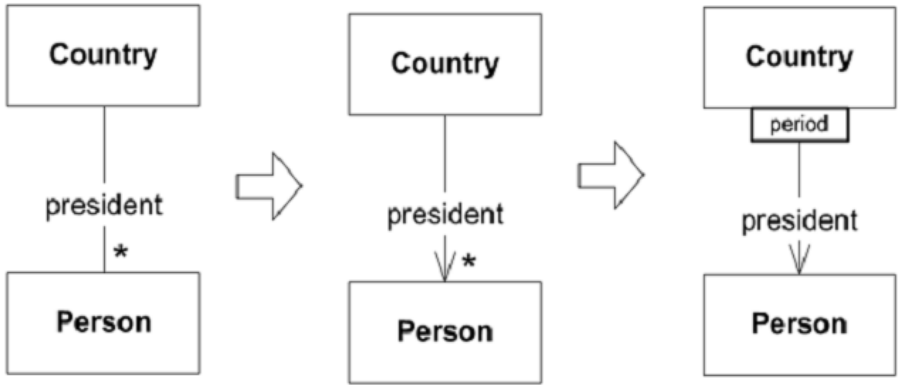
Java class skeleton

Class and Object Diagrams

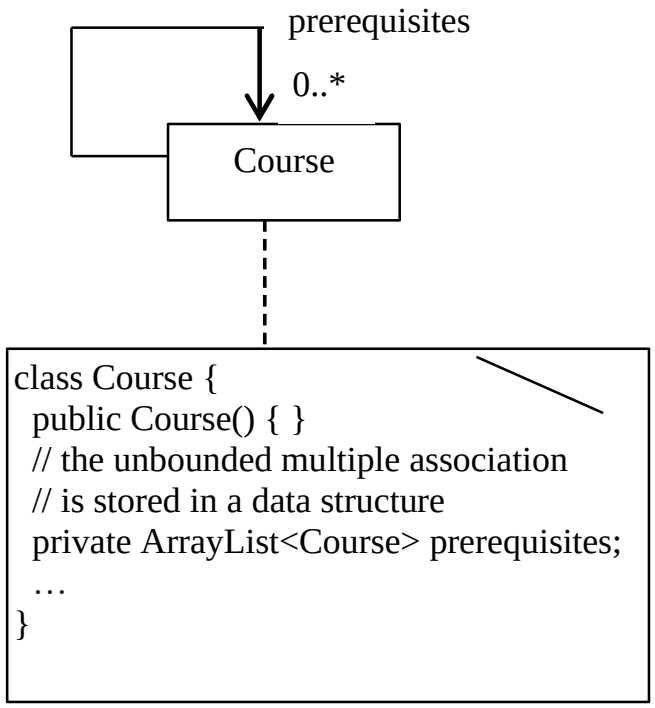
- Associations – Special Cases



Association as a class



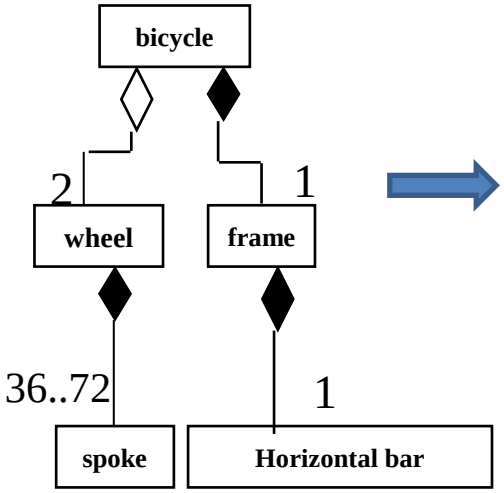
a) **Association** b) **Association traversal** c) **Qualified association**



Reflexive associations (hierarchical or network structures)

Class and Object Diagrams

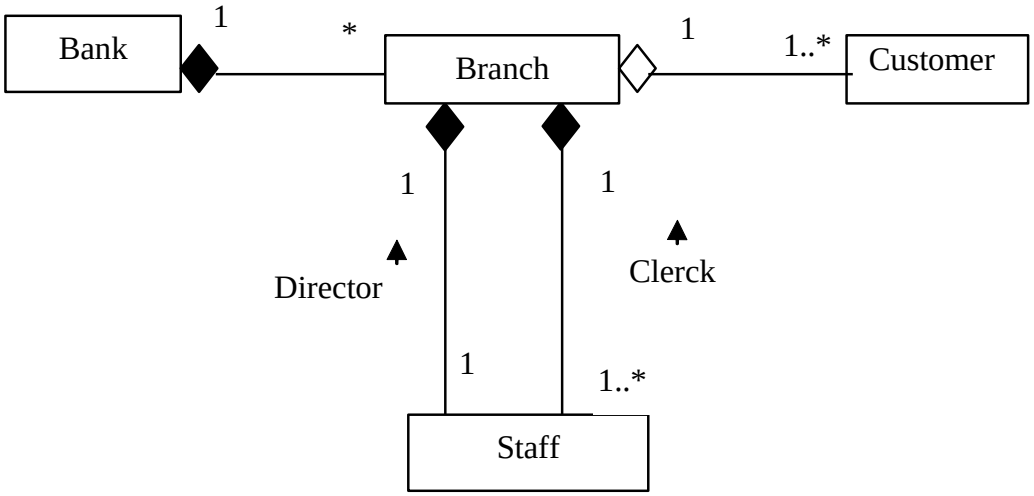
- Aggregation
  - Special form of association modeling a **“whole/part”** relationship
  - Represents a “**has – a**” or “**part – of**” relationship
  - Types: weak and strong (i.e., composition)
  - Properties: transitive, not reflexive
  - Responsibility in creating and destroying the parts



Whole	Part	Aggregation type	Motivation
Bicycle	Wheel	Weak aggregation	The bicycle and wheels can exist independently – we can add other wheels or add the wheels to another bicycle.
Bicycle	Frame	Composition	The parts have coincident lifetime with the whole – if the whole is destroyed then the parts are destroyed as well. • A frame belongs to exactly one bicycle; when you create a frame for a bicycle you must attach it to the bicycle. • Spokes belong to exactly one wheel; when you create spokes for a wheel you must attach them to the wheel. • A horizontal bar belongs to exactly one frame; when you create a horizontal bar for a frame you must attach it to the frame.
Wheel	Spoke		
Frame	Horizontal bar		

Class and Object Diagrams

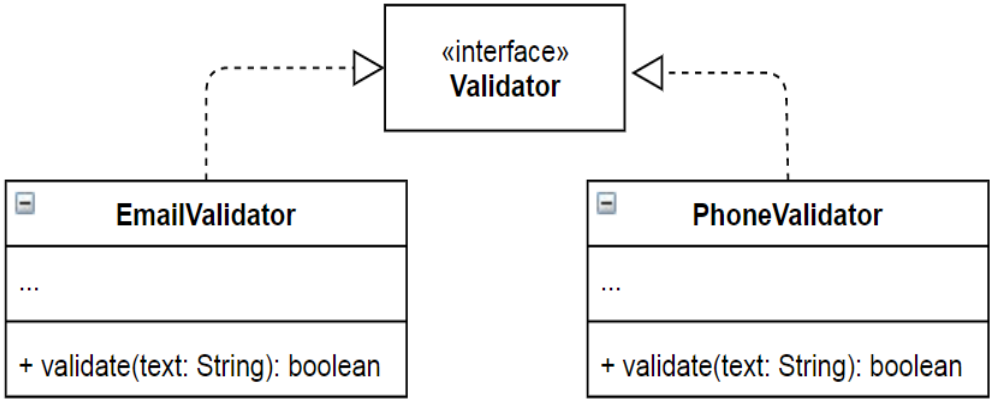
- Aggregation - Example



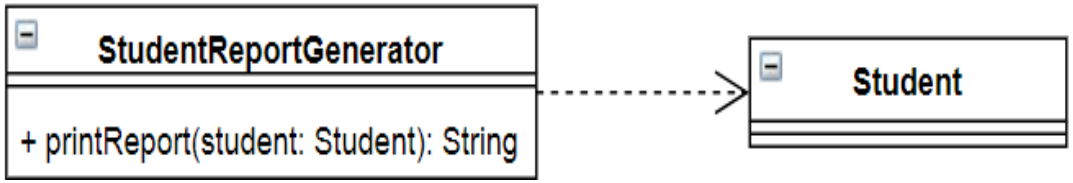
Whole	Part	Aggregation type	Motivation
Bank	Branch	Composition	The bank consists of several branches – the branches will no longer exist if the bank closes.
Branch	Staff	Composition	The staff of a branch will no longer be the employees of the branch when it will no longer exist. *However, the composition is debatable and depends on the interpretation: if a branch will be closed, the employees will still exist and would be assigned to another branch => we could use aggregation instead of composition.*
Branch	Customer	Weak aggregation	The customer of a branch will still be customers in case the branch closes and could be assigned to another branch.

Class and Object Diagrams
Realization and Dependency

- Realization



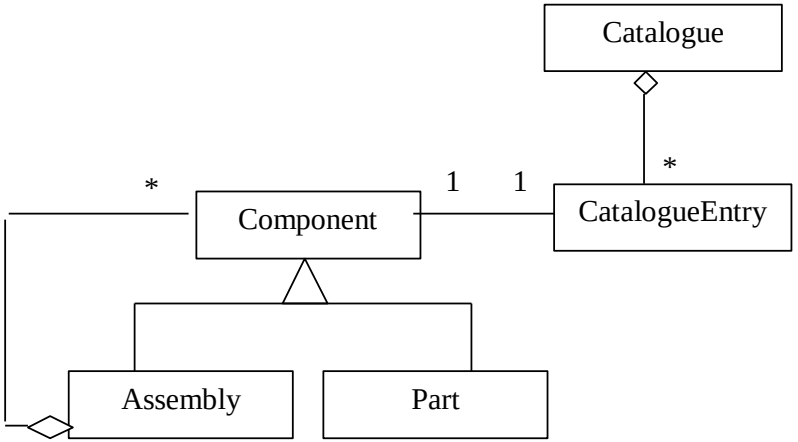
- Dependency



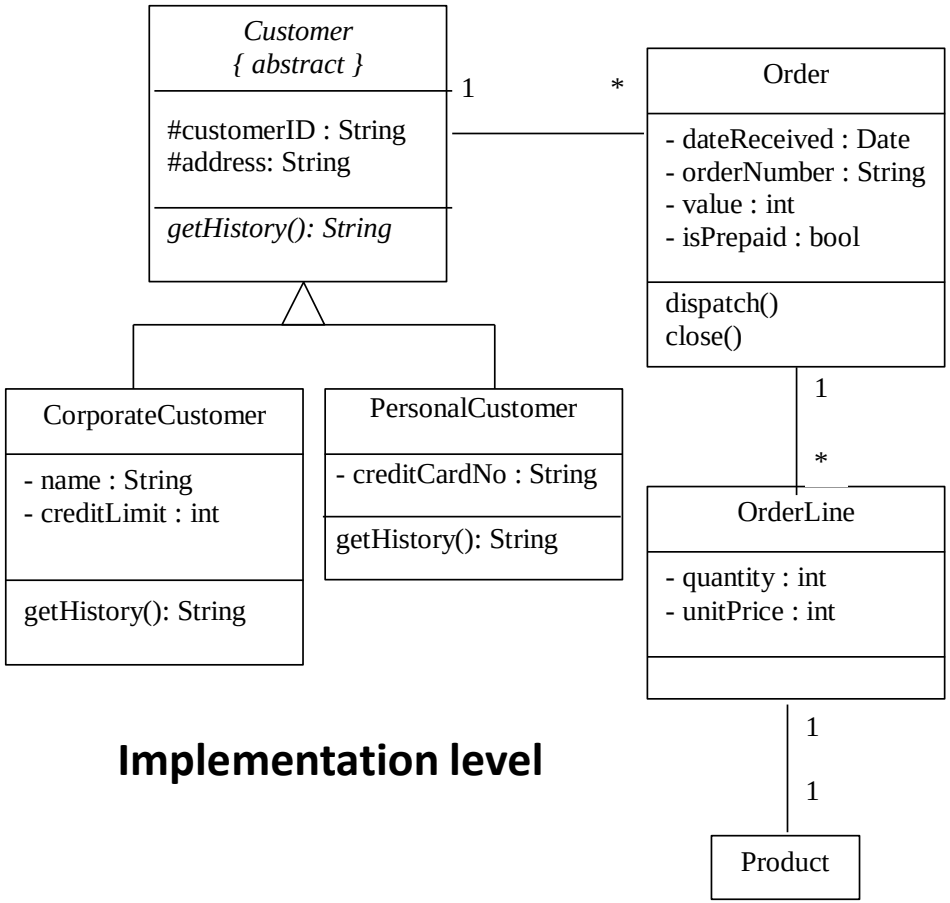
- Minimize dependencies and favor loosely coupled designs
- Improve the flexibility and maintainability of software systems.

Class and Object Diagrams

- Examples



Conceptual level

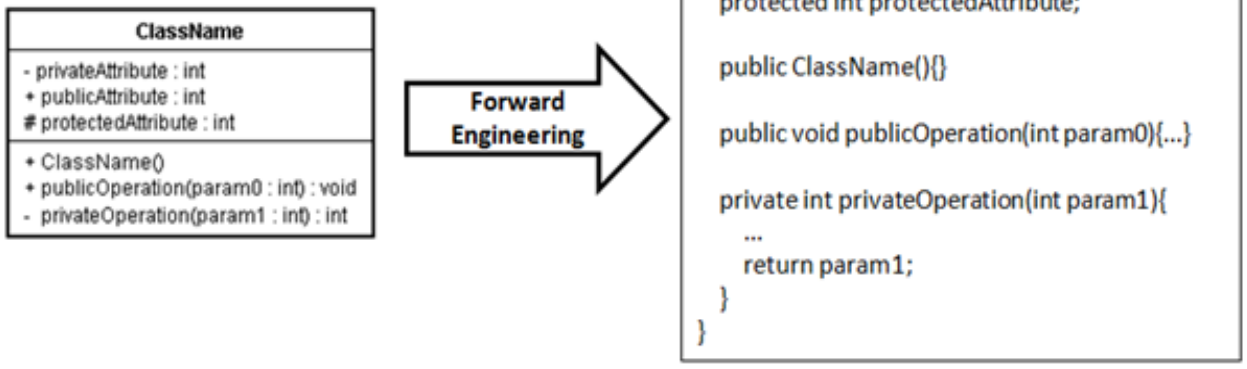


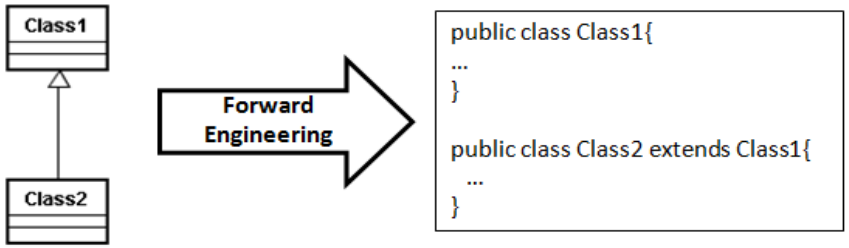
Implementation level

Adapted from: M.Fowler–
UML Distilled, Addison Wesley 2000

Class and Object Diagrams

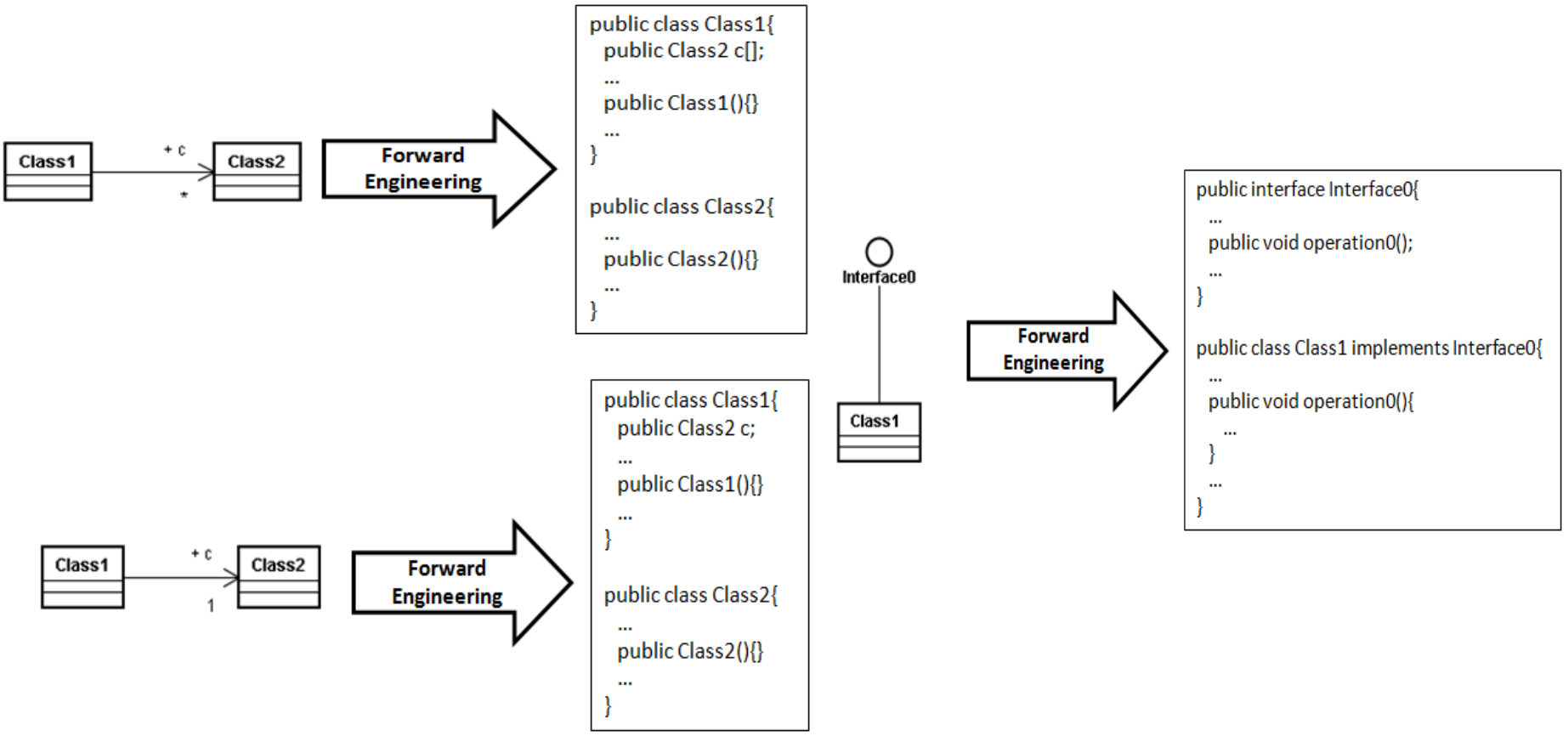
- Forward Engineering
  - Create detailed blueprint design for developer





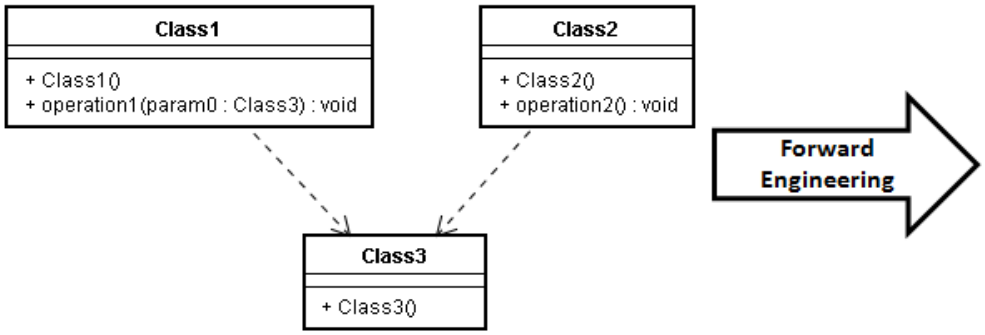
Class and Object Diagrams

- Forward Engineering - Examples



Class and Object Diagrams

- Forward Engineering - Examples



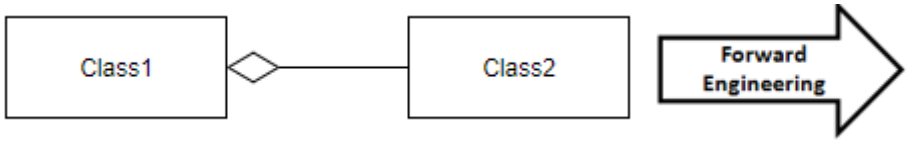

```
class Class1 {
   ...
   public Class1() {...}
   ...
   public void operation1(Class3 param0) {...}
   ...
}

public class Class2 {
   ...
   public Class2() {...}
   ...
   public void operation2()
   {
      ...
      Class3 c = new Class3();
      ...
    }
   ...
}

public class Class3{
  ...
   public Class3() {...}
   ...
}
```

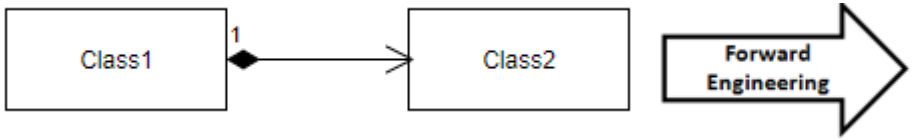
Class and Object Diagrams

- Forward Engineering - Examples




```
public class Class1 {
   private Class2 class2Object;
   …
   public Class1(Class2 class2Object){
      this.class2Object = class2Object;
   }
    …
}
```

Class1 is not responsible for creating the Class2 object;
The object is created outside Class1 and it is passed as an argument to the constructor of Class1



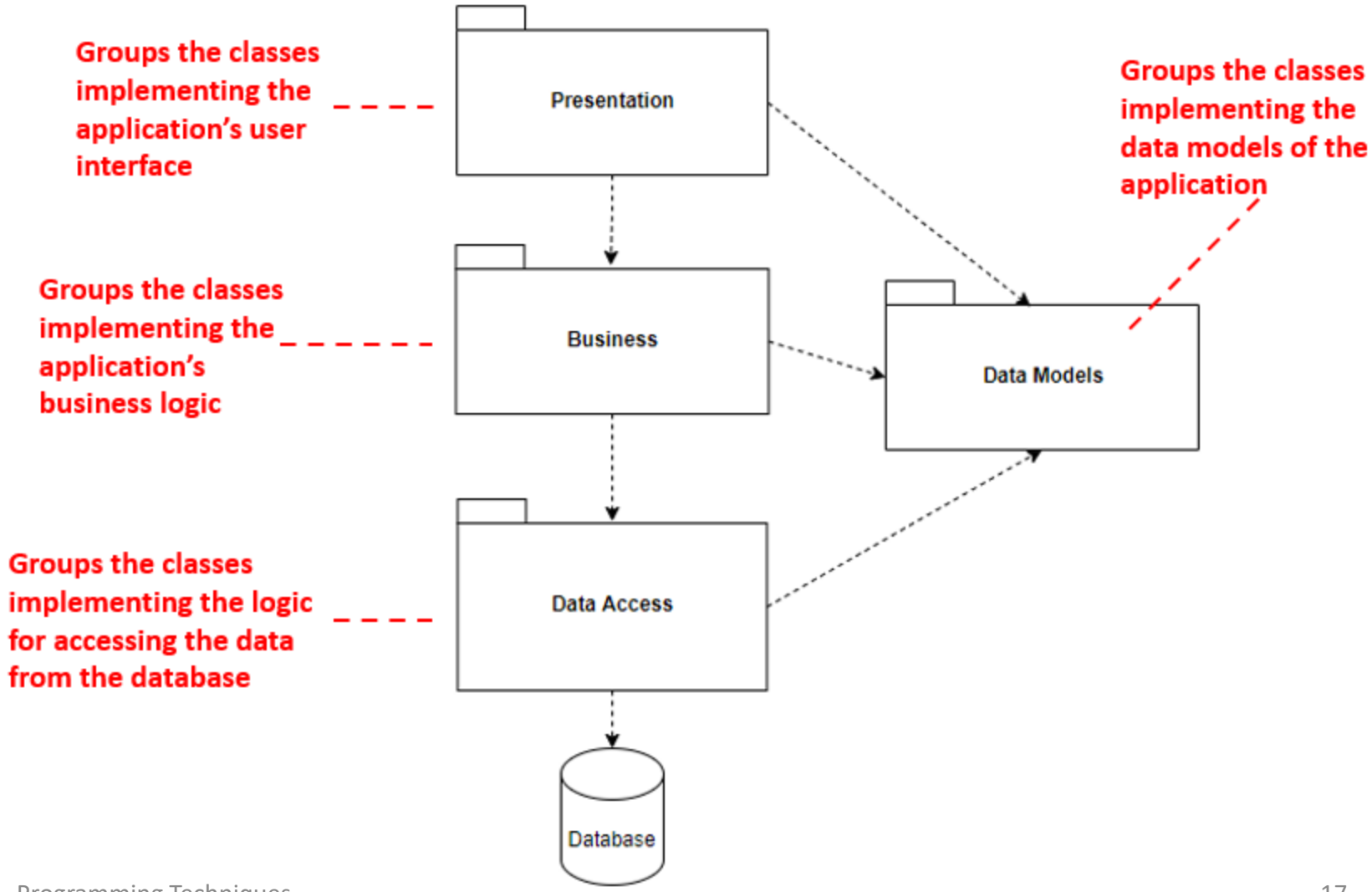

```
public class Class1 {
   private Class2 class2Object;
   …
   public Class1(){
      this.class2Object = new Class2();
   }
    …
}
```

Class1 is responsible for creating the Class2 object;
The object is created when a Class1 object is created.

Package Diagrams

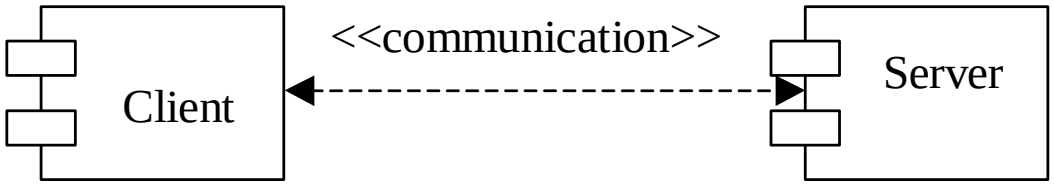
- Illustrate packages of classes and their dependencies
- Useful in decomposing large systems into subsystems
- UML package
  - Collection of modelling elements that are grouped together because they are logically related
- Should apply the principles of high cohesion and low coupling

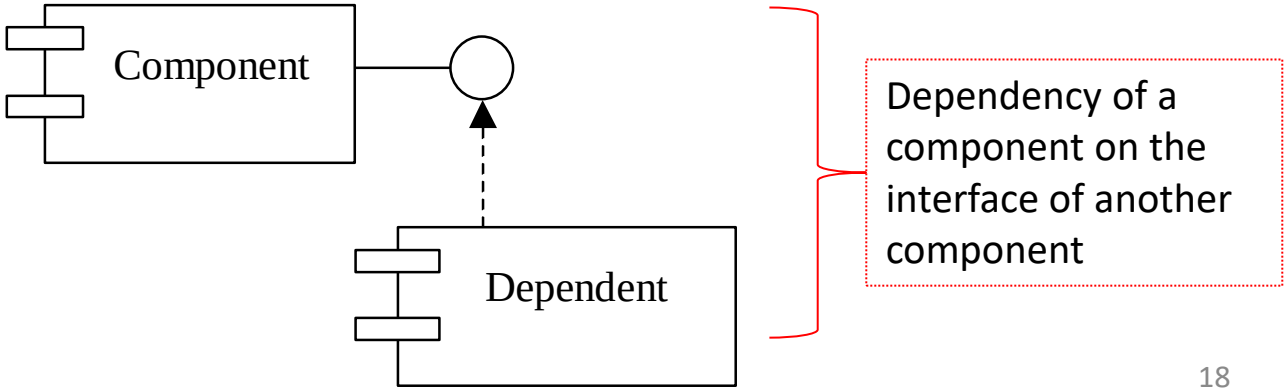
Package Diagrams



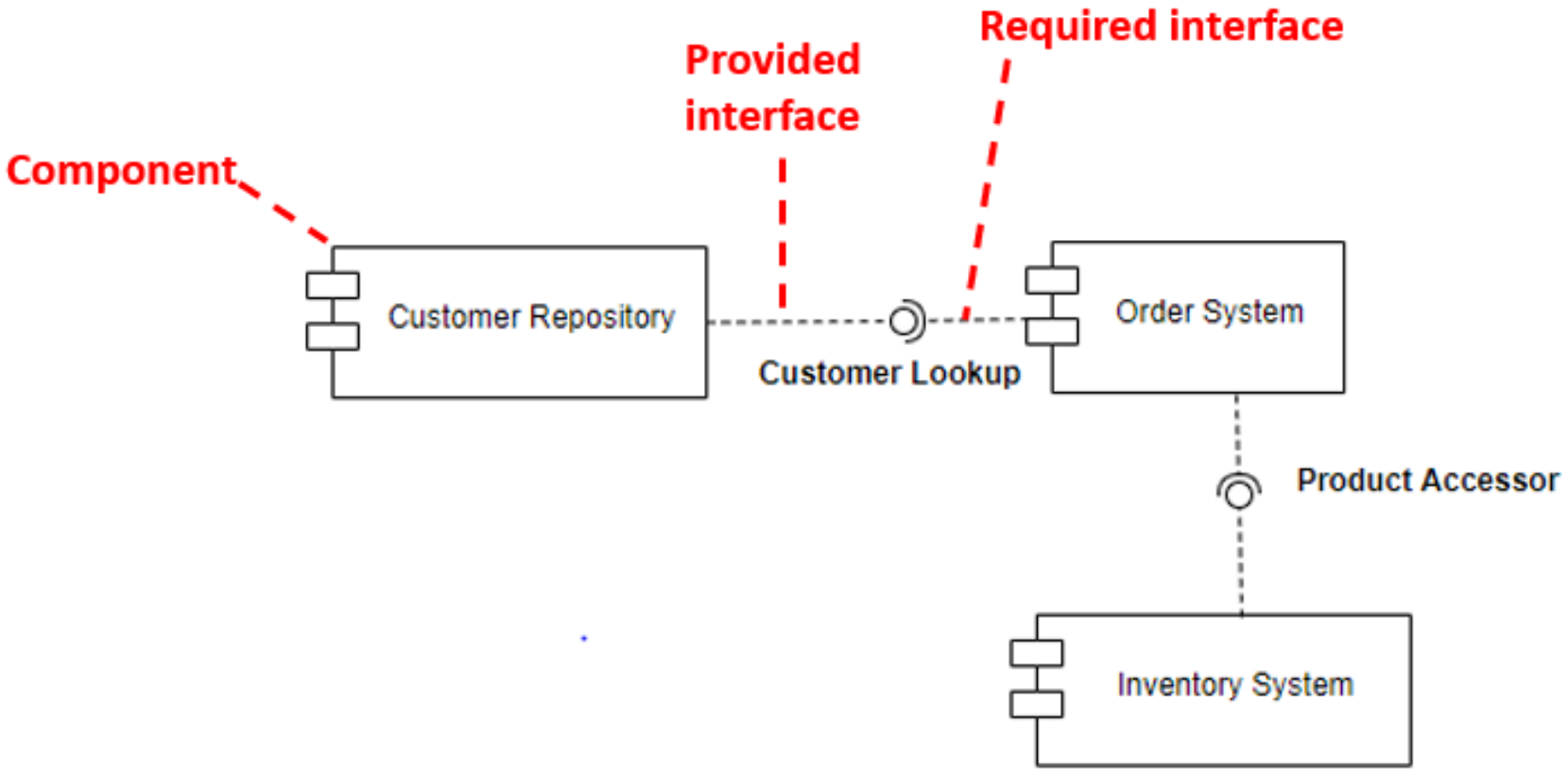
Component Diagrams

- Illustrate the physical structure of a software system,
  - how system components relate to each other
- Used with deployment diagrams to show the physical location of components of the system





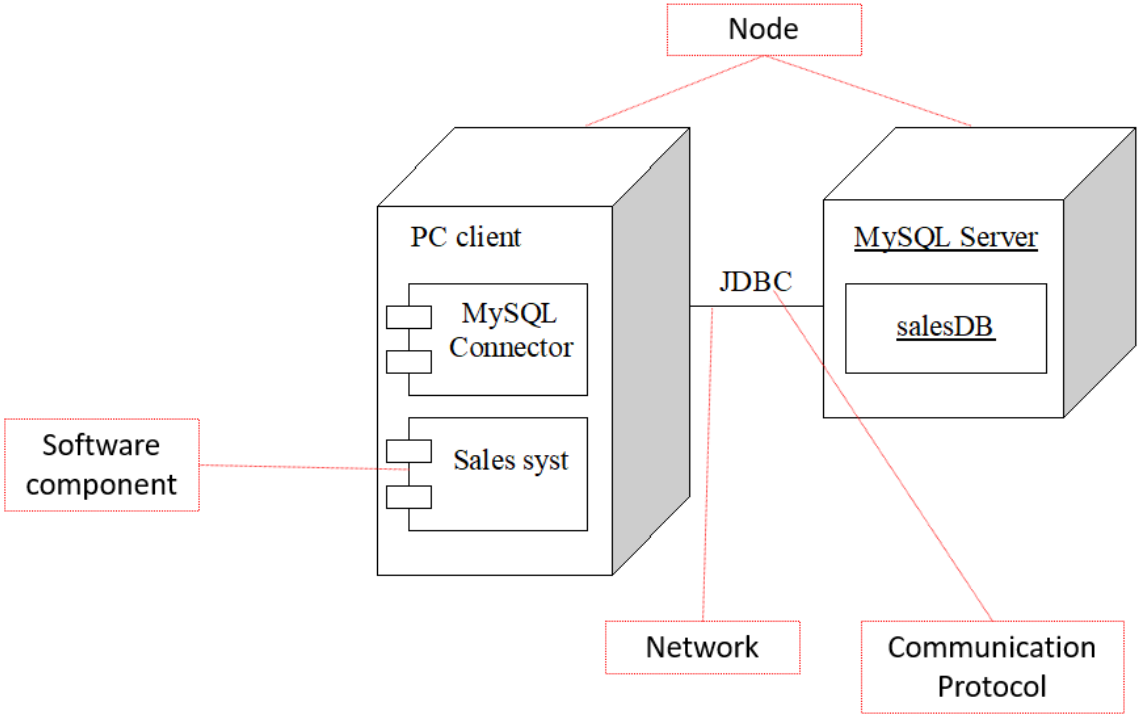
Component Diagrams



Adapted from Link

Deployment Diagrams

- Show the mapping of software to hardware configurations (i.e., the physical architecture of the system)
  - Configuration of run-time processing elements and the software components and processes located on them



Use Case Diagrams

- Use-cases
  - Set of scenarios related to how the system is used
  - Help to discover
    - System entities
    - System actors (roles)
    - Attributes
    - Behavior
    - How actors interact with system resources
- Use case diagrams
  - Graphically capture system actors, use-cases and their relationships => requirements
- Use-case document
  - Describes the sequence of actor - system interaction

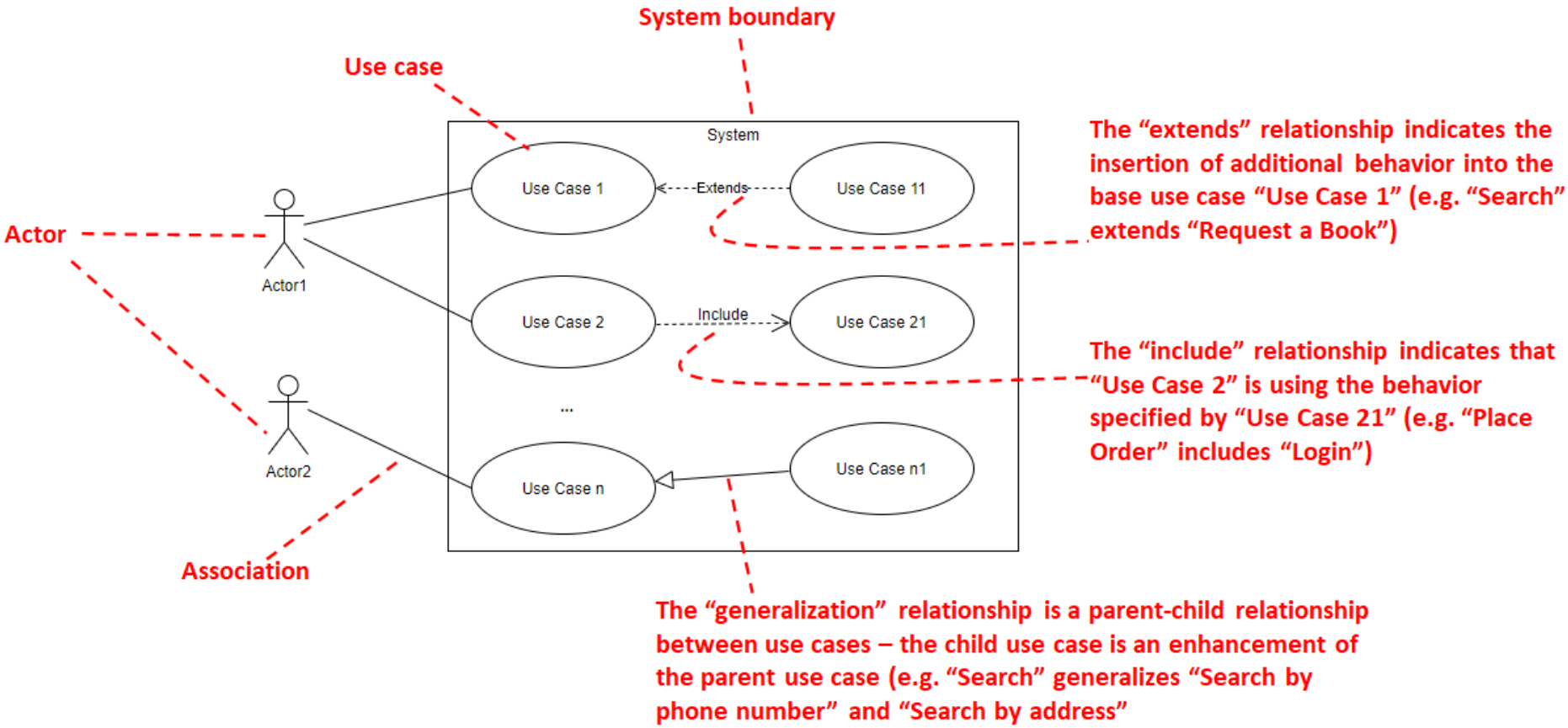
Use Case Diagrams

- Use-Cases Document Example

Use Case <S#.#>: <Use Case Name (best use a verb of action)>	
Brief Description	<A brief description of the use case, no longer than a small paragraph; preferably keep it one clear goal per use case>
Parent Scenario	<Scenario number>
Actor(s)	<All users interacting with the System; preferably identified by role name>
Priority	<Depending on how many use cases we end up with we may need to prioritize them based on importance to meeting the project's objective: high (must do) | medium (should do, time permitted) | low (nice to do, but most likely not)>
Trigger	<Concrete actions by users interacting with the System to initiate use case>
Pre-conditions	<System State prior to launching the use case >
Post-conditions	<System State expected after the use case has been completed >
Basic Flow	<Describe the basic flow of events during use case Step 1. Step 2. … Step N. >
Alternate Flow(s)	<If applicable, provide the steps of less common user/system interactions>
Exception Flow(s)	< Anything that may happen that would prevent the user from achieving their goal >

Use Case Diagrams

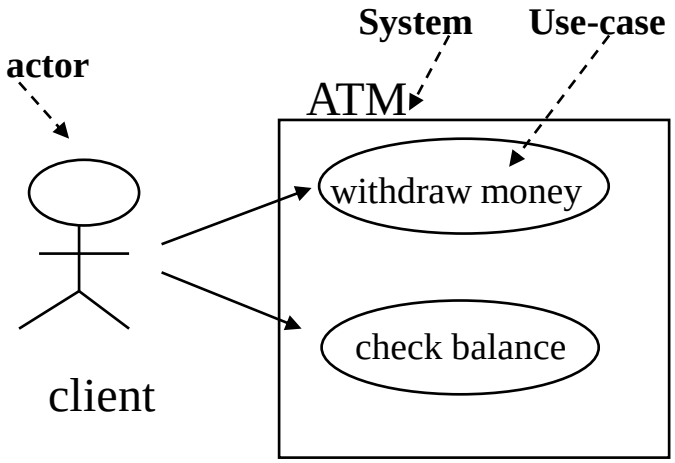
- Graphical representation



Adapted from Link

Use Case Diagrams

- Example - ATM



- **Use case description example**

Use Case: withdraw money
Primary Actor: Client
Success Scenario Steps:

1. The client introduces the card in the ATM
2. The ATM checks if the card is valid
3. The ATM requests the client to insert the pin
4. The client introduces the pin
5. The ATM checks if the date is correct
6. The ATM requests the client to specify the amount of money
7. The client introduces the amount of money
8. The ATM checks if the amount of money is under the daily limit
9. The ATM asks the client if he wants a receipt
10. The client specifies that he wants the receipt
11. The ATM returns the card
12. The client takes the card
13. The ATM returns the money and the receipt
14. The client takes the money and the receipt

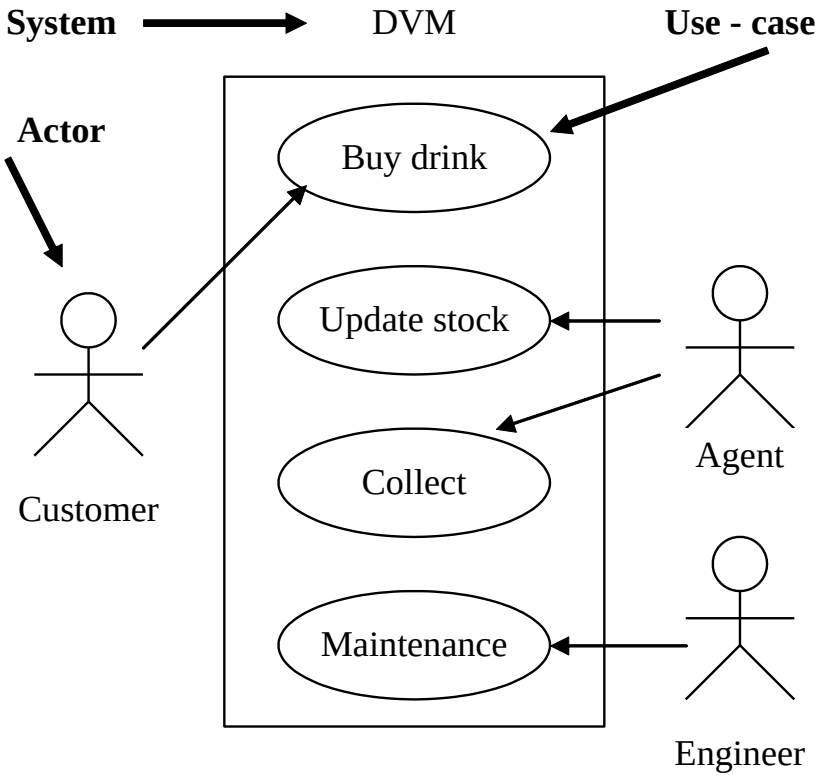
Alternative Sequences:

a) Incorrect pin
- The ATM tells the client that the pin is incorrect
- The scenario returns to step 3

b) The amount of money is higher that the daily limit
- The ATM tells the client that the amount of money is higher that the daily limit
- The scenario returns to step 7

Use Case Diagrams

- Example – Vending Machine



Sequence and Activity Diagrams

Illustrate the system behaviour

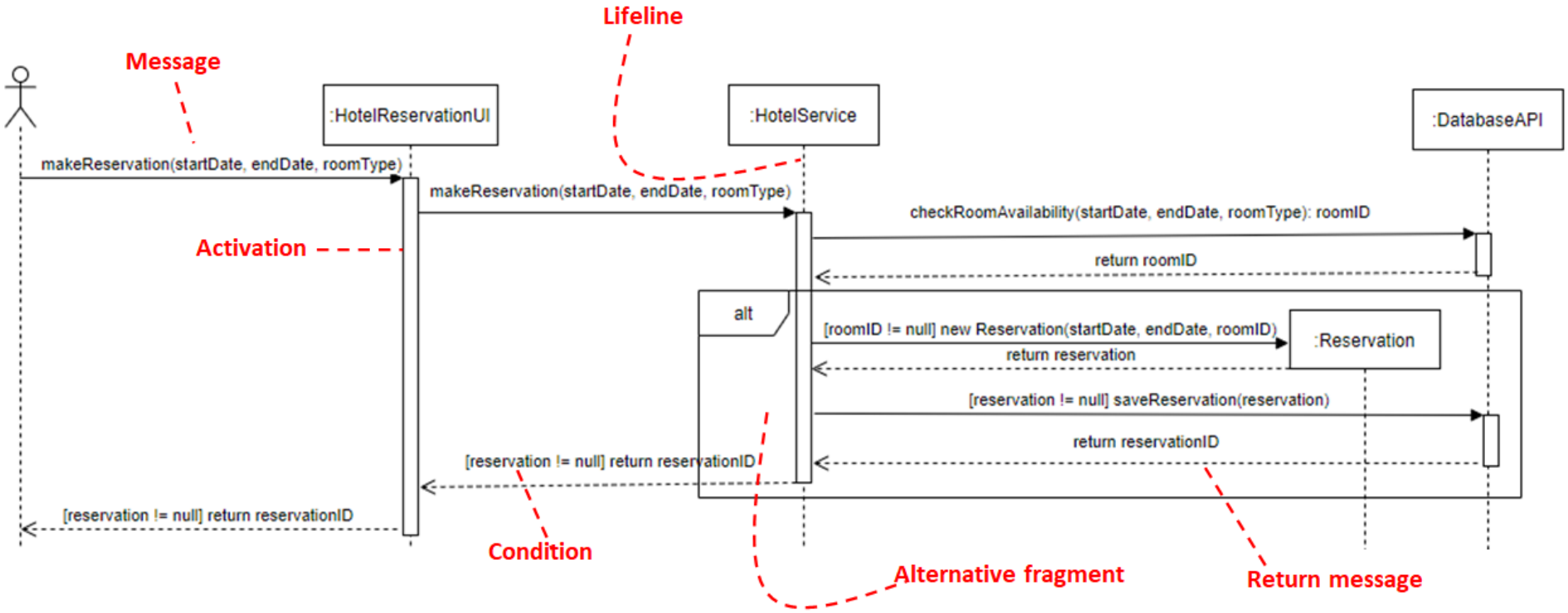
- Interaction diagrams

Typically capture the behaviour of a single use-case

- Show the objects involved and the messages passed among the objects
- Show the collaboration among the objects

Sequence Diagrams

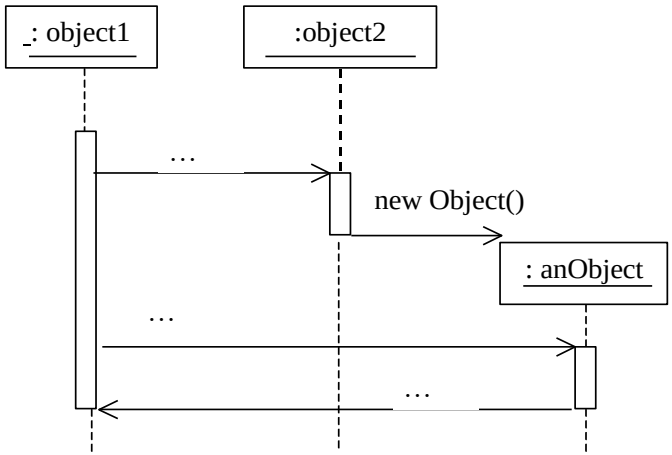
- Show the objects involved in the interaction



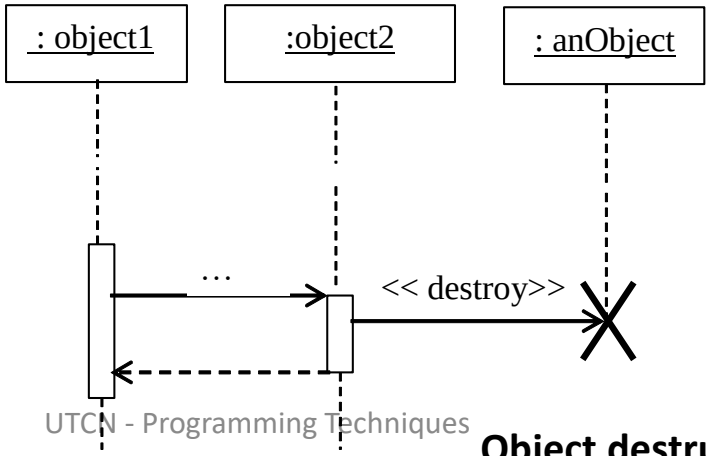
Adapted from Link

Sequence Diagrams

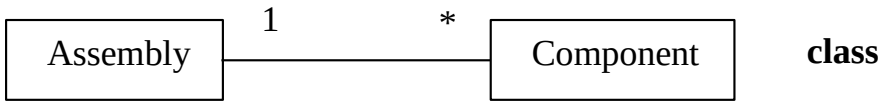
- Special constructs



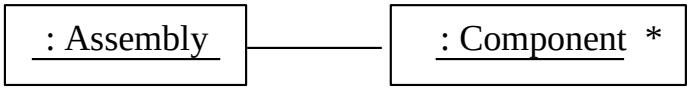
Object creation



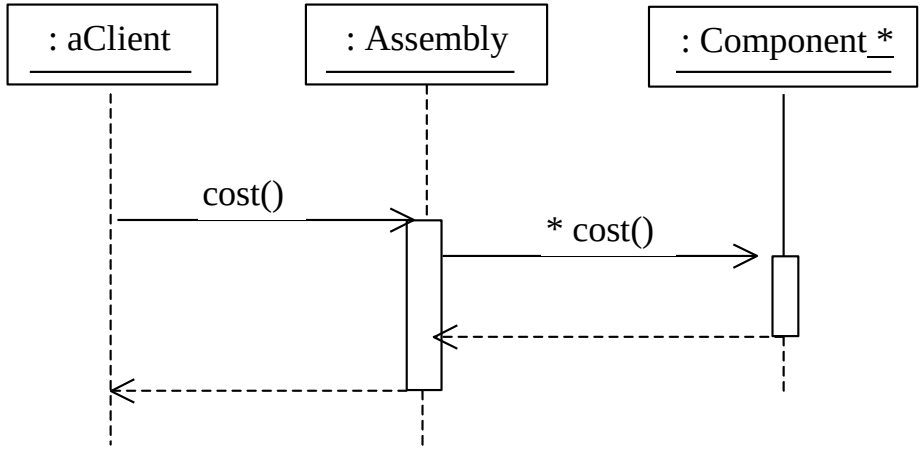
Object destruction



class



objects

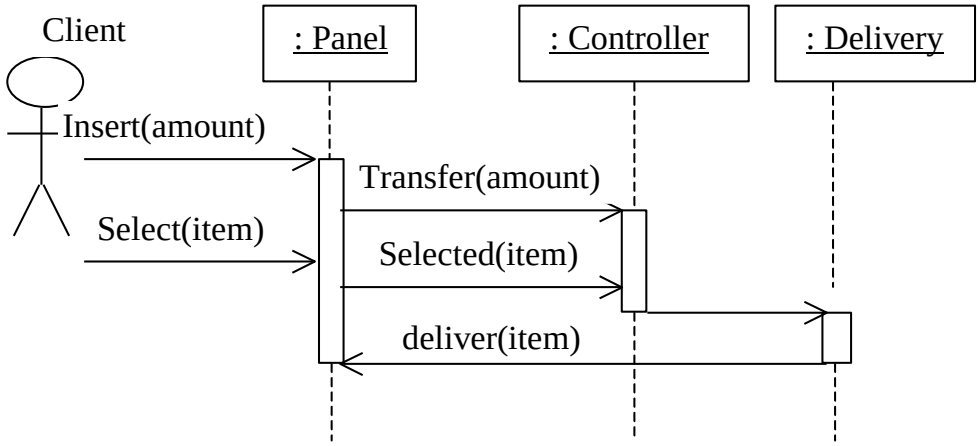


Iterated messages

Sequence Diagrams

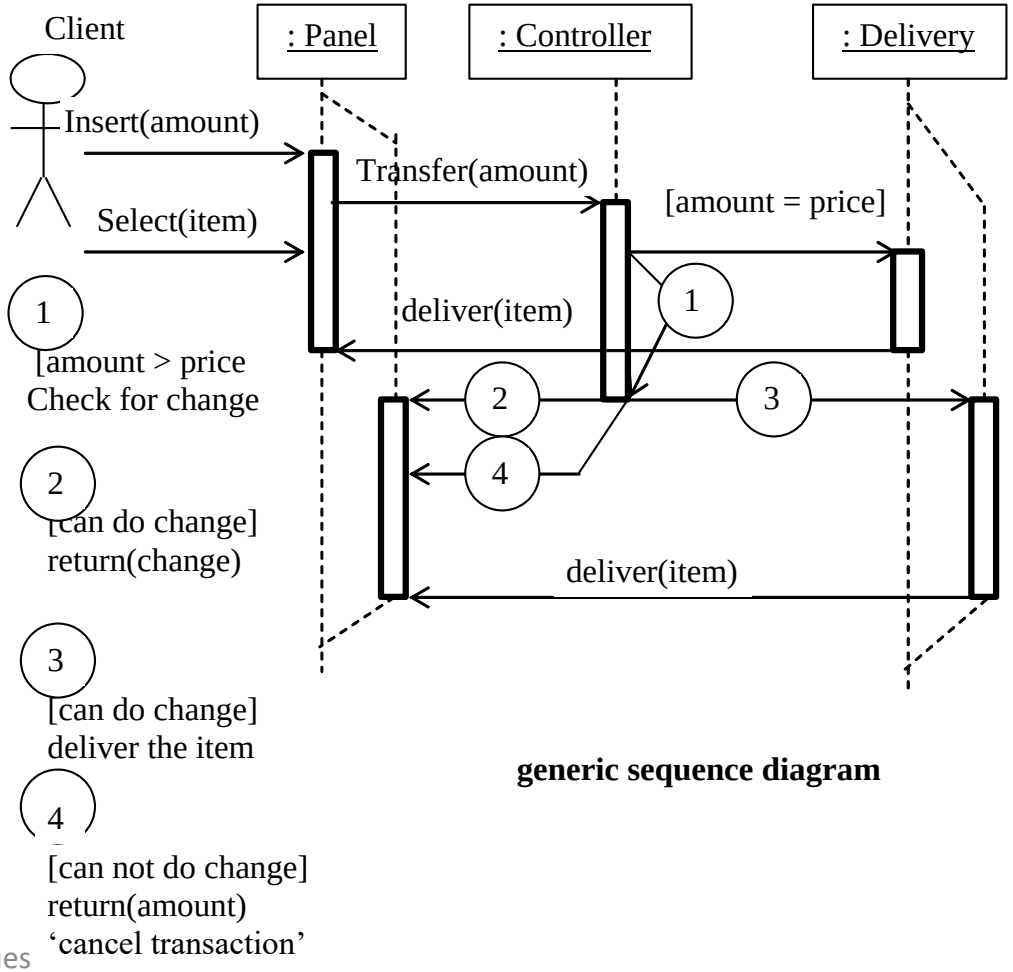
- Example for the “Buy drink” use case – simple case

1. Client inserts the money into machine front panel
2. The client makes a selection
3. The money go to the controller
4. The controller checks if the selected item is in the store unit
5. Assuming a best-case scenario, the item exists, the controller updates the cash and items and ask the delivery unit to deliver the item from the store
6. The deliver unit delivers the items in the front of the machine



Sequence Diagrams

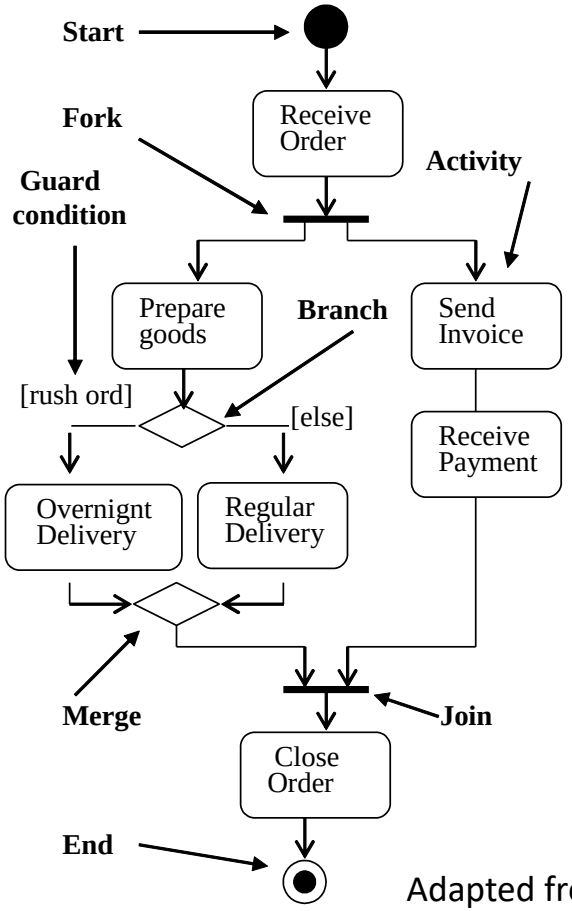
- Example for the “Incorrect amount of money’” use case

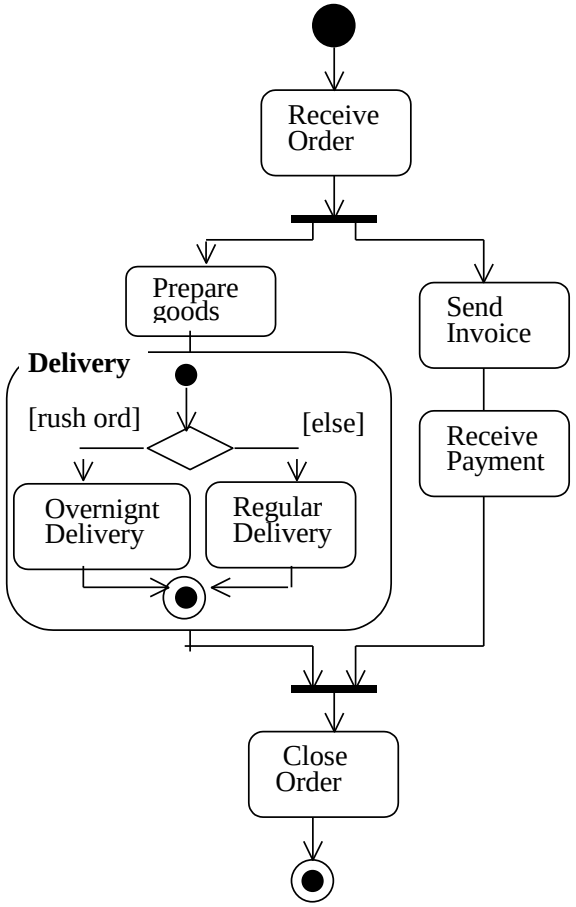


generic sequence diagram

Activity Diagrams

- Illustrate the flow of events in a use case

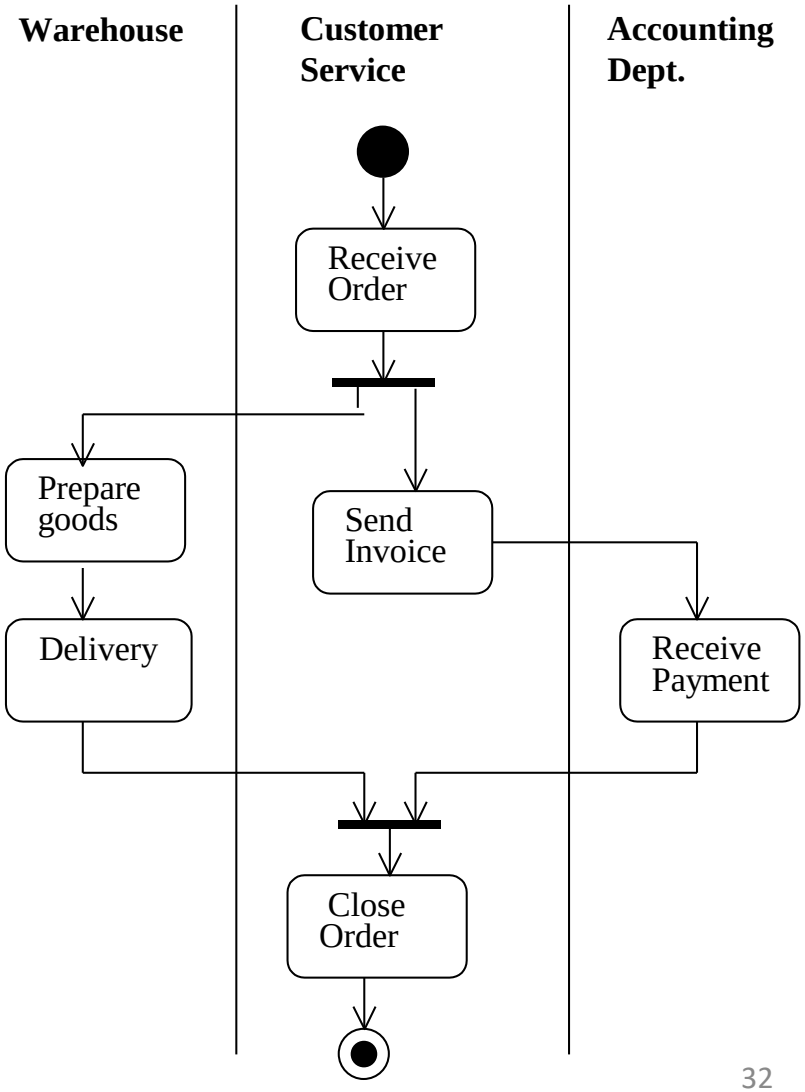




Adapted from: M.Fowler–
UML Distilled, Addison Wesley 2000

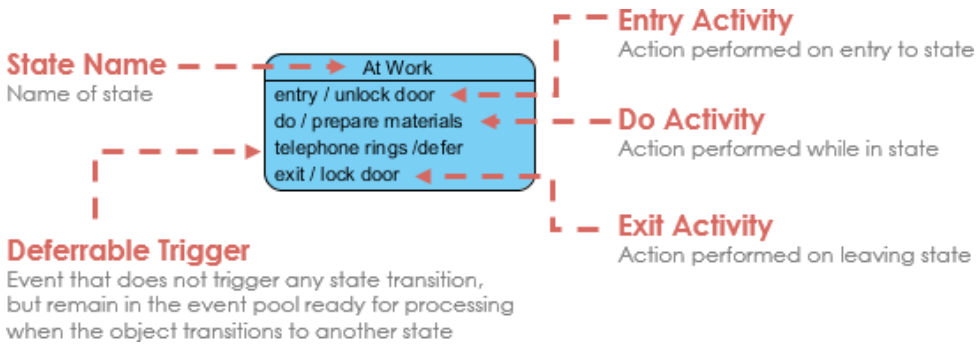
Activity Diagrams

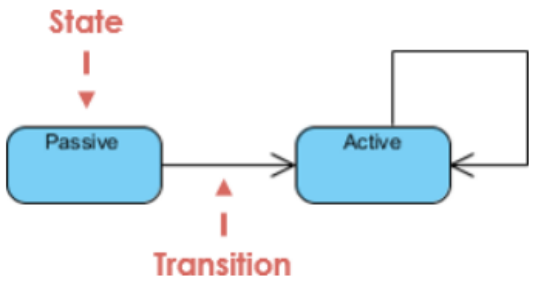
- Activities and actions can be grouped into conceptual swim lanes
  - Activities in each lane can be associated to a department (part) of an organization

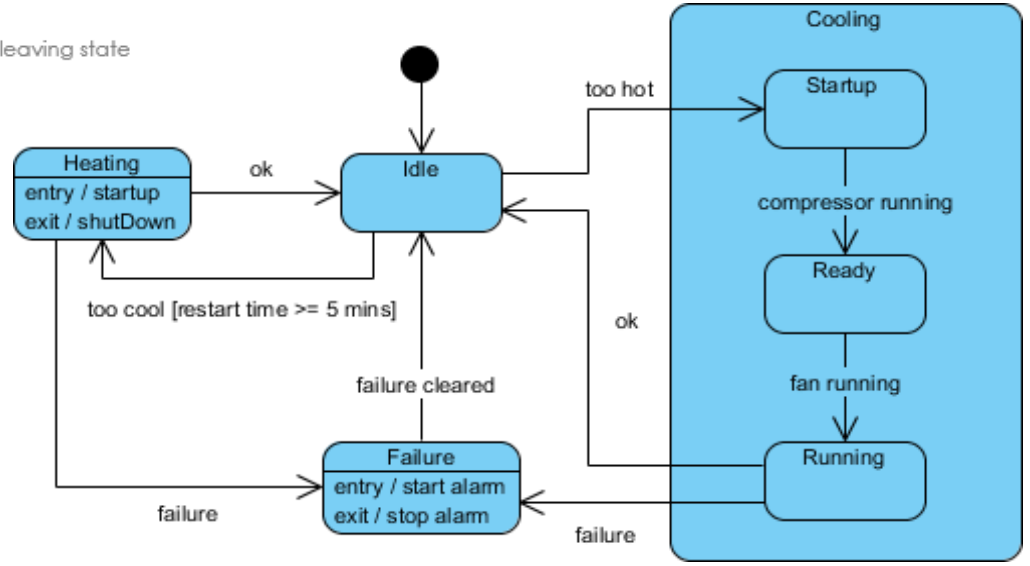


State Machine Diagrams

- Describe state-dependent behavior for an object
- Usually applied to objects
  - Can be applied to any element that has behavior to other entities (e.g., actors, use cases, methods, subsystems)





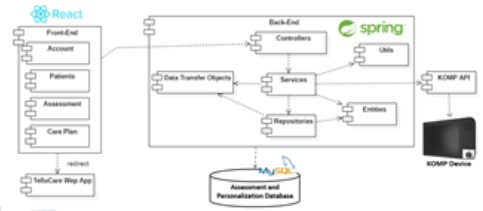


Source

“4+1” View Model of Software Systems

UML Class Diagrams

UML Component Diagrams



End-user
Functionality

Programmers
Software management

Logical View

Development View

Use-cases View

Process View

Physical View

Integrators
Performance
Scalability

System engineers
Topology
Communications

UML Sequence Diagrams

UML Deployment Diagrams