

Software engineering

Introduction

- Contact:
- zoran.sverko@famnit.upr.si

What is Software Engineering?

- IEEE definition (1993):
 - The application of a systematic, disciplines, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software.

What is Software Engineering?

- „Software engineering is the study or practice of using computers and computing technology to solve real-world problems. Computer scientists study the structure, interactions and theory of computers and their functions. Software engineering is a part of computer science in that software engineers use the results of studies to build tools and techniques to meet the needs of customers.” [1]

Why do we need software engineering?

'Life Story of a Program' [2]

- 1. The programmer has written and submitted a program that he believes is bug-free.
- 2. The program was tested and 20 bugs were found.
- 3. The programmer corrects 10 errors and tells the test team that the remaining 10 errors are not real 'bugs'.
- 4. The test team finds that 5 corrections do not work and discovers 15 new bugs.
- 5. Until the sales and marketing people lose patience, the following story is repeated:

Read point 3.

Read point 4.

[2]

- 6. Due to business and marketing pressure (the announcement of a new product was based on a far too optimistic project plan), the program goes live.
- 7. Users find 137 new bugs.
- 8. The original programmer - the author of the program (having already received payment for his 'invention') is nowhere to be found.
- 9. The newly assembled development team fixes almost all 137 bugs, but finds 468 new bugs.

[2]

- 10. The original programmer sends a poorly paid test team a postcard from the Maldives and the Dominican Republic. The entire testing team has given up.
- 11. A hostile takeover of the company by a rival firm occurs. The new owners collect all profits from the last version of the program, which contained 783 bugs.
- 12. The new chief information officer (CIO) joins the board of directors. He hires a programmer to create a new program from the remains of the old program.

[2]

- 13. The programmer writes a program that he believes is bug-free and submits it.
 - 14. Return to point 2.
-
- What do you think about the program's resume? Is the story just fictional or do such cases exist in practice?
 - Unfortunately, we also find such or very similar cases all too often in practice. The quality of the software is always questionable.

Engineering failures

- Let us start with failure in the traditional engineering categories.
- The first Tacoma Narrows Bridge opened to traffic on July 1, 1940. Its main span collapsed into the Tacoma Narrows four months later on November 7, 1940, at 11:00 a.m. (Pacific time) as a result of **aeroelastic flutter caused** by a 42 mph (68 km/h) wind. The bridge collapse had lasting effects on science and engineering.
- https://en.wikipedia.org/wiki/Tacoma_Narrows_Bridge
- Cost: \$6 Million in 1940 - now almost \$1 Billion
- the wrong calculation

SW products and failures

- Software products do include defects.
- All is related to three limitations of SW engineering: cost, time and product (product must satisfy the customer).
- Software engineering offers techniques to reduce the number and fatality of defects/bugs.
- But we can (never?) avoid them...
 - *“...building software will always be hard. There is inherently no silver bullet.”* F. Brooks
- Furthermore, we will see some famous mistakes in software engineering.

Examples of software engineering failures

- **EDS Child Support System**
- In 2004, EDS introduced a highly complex IT system to the U.K.'s Child Support Agency (CSA). At the exact same time, the Department for Work and Pensions (DWP) decided to restructure the entire agency. The two pieces of software were completely incompatible, and irreversible errors were introduced as a result. The system somehow managed to overpay 1.9 million people, underpay another 700,000, had US\$7 billion in uncollected child support payments, a backlog of 239,000 cases, 36,000 new cases "stuck" in the system, and has cost the UK taxpayers over US\$1 billion to date.
- <https://raygun.com/blog/costly-software-errors-history/>

- **Soviet Gas Pipeline Explosion**
- The Soviet pipeline had a level of complexity that would require advanced automated control software. The CIA was tipped off to the Soviet intentions to steal the control system's plans. Working with the Canadian firm that designed the pipeline control software, the CIA had the designers deliberately create flaws in the programming so that the Soviets would receive a compromised program. It is claimed that in June 1982, flaws in the stolen software led to a massive explosion along part of the pipeline, causing the largest non-nuclear explosion in the planet's history.
- <https://raygun.com/blog/costly-software-errors-history/>

- **Heathrow Terminal 5 Opening**

- Just before the opening of Heathrow's Terminal 5 in the UK, staff tested the brand new baggage handling system built to carry the vast amounts of luggage checked in each day. Engineers tested the system thoroughly before opening the Terminal to the public with over 12,000 test pieces of luggage. It worked flawlessly on all test runs only to find on the Terminal's opening day the system simply could not cope. It is thought that "real life" scenarios such as removing a bag from the system manually when a passenger had left an important item in their luggage, had caused the entire system to become confused and shut down.
- Over the following 10 days some 42,000 bags failed to travel with their owners, and over 500 flights were cancelled.
- <https://raygun.com/blog/costly-software-errors-history/>

Success of software products

- Success rate of software products is lower than success rate in other engineering disciplines!
- SW product can be successful if
 - The development is completed
 - Is useful
 - Is usable
 - Is used - At the end this is the only criterion, nothing else. This is different to company success, where finance perspective is crucial !!!

Reasons for failure

- Quality issues – the system is faulty (bugs)
- Development does not meet deadlines
- Over budget (too expensive)
- Does not fulfill user needs.
- Difficult to maintain.

Assignment !

- Please open the computers in front of you and find your own examples of software engineering fails. You have 10 minutes to search!
- Afterwards, some of you will present examples to the others.

Characteristics of good SW products

- **Functionality** – to fulfill user needs
- **Maintainability** – ability to further develop SW due to changed requirements.
- **Reliability** – SW worth to trust.
- **Efficiency** – usage of system resources.
- **Acceptability** – ability to be accepted by users for who it was developed: understandable, usefull, compatible with other systems.

Engineering principles

- Following and managing the project progress
- Designing of quality control
- Exhaustive testing

Software process phases

- Problem definition
- Feasibility study
- Requirements analysis
- System modeling
- Component modeling
- Implementation
- Testing
- Deployment*
- Maintenance

Project

- We will be developing a new SW product.
- The emphasis is on the development approach.
- Assessment of the products as results of each SW development phase except the implementation phase (documents) and contribution to the team.

More about projects

- 3 students in each team (group), in a special case 2/4.
- Projects follow the SW development phases.
- Each student prepares his/her own proposal for each of the phases.
- Each phase ends with coordination in which the team selects and proposes a solution based on »best« member proposals.
- Only the implementation of the first project phases is required and assessed (ending with the component design).
- Actual implementation of a SW product is not required.
- Collaborate!

Project assessment

- Quality of products in each of the life cycle phases (student and group version).
- Contribution of an individual to the team (group).

Week nr.	Date	Tutorials	Homework asignment
1	01.10.21	-	
2	04.10.21	Introduction in course and the need of SW engineering, formation of project groups, examples of projects, Examples and explanation of the problem definition phase	Project problem definition (individual).
3	11.10.21	Feasibility study, an example of searching for problem solutions, defining requirements, defining user stories.	Feasibility study for solving your problem, existing alternatives (individual).
4	18.10.21	Modeling and diagramming (UML and others). Working environments (diagramming tools, revision control systems)	Groups propose/select their topic and find an agreement on the general solution direction.
5	25.10.21	UML use case diagrams.	Use case diagrams (individually)
6	01.11.21	UML activity diagrams and UML sequence diagrams.	Use case diagrams and use case descriptions (individually)
7	08.11.21	UML state machine diagrams	(groups) Complete Software requirements specifications.
8	15.11.21	UML component diagrams and UML deployment diagrams	Submission of specifications review reports (individual – for other group)
9	22.11.21	Midterm exam 1	
10	29.11.21	System design: specification of components and interfaces	System design (individual)
11	06.12.21	Component design using UML class diagrams and UML object diagrams	System design (group agreement).

12	13.12.21	Testing: approaches, code reviews, white box testing based on cyclomatic complexity and basic paths, conditions and loops.	System design review (individual for other groups)
13	20.12.21	Testing: black box testing: Equivalence partitioning and boundary value analysis.	Testing scenario for a selected use case (groups)
14	27.12.21	-	
15	03.01.22	Testing – more examples	Review of testing scenarios for other groups.
16	10.01.22	2 nd midterm exam	
17	17.01.22	Overview and conclusions of the course, responsibilities of a software engineer	

Problem definition:

- Questions answered:
 - **What is a problem?**

This is the 1st phase.

Example: „Appropriate clothing size”

- Problem statement:

People face various problems while choosing the right size of clothes in a web shop:

- Sometimes they are unsure about the choice because they do not have enough information about the clothes.
- What size of clothes do other people with similar body size choose?
- There are too many suggestions and the most important ones are overlooked?

The goal would be to increase the selection of the right dress size and reduce the number of returns.

Could this be solved by a software product – Appropriate clothing size?

Problem solution:

- **“A computer system for...” is a solution and not problem!**

Some possible solutions:

- Let each clothing manufacturer enter the exact dimensions for their product. Have each garment manufacturer enter the exact measurements for their product (garment width, garment height, sleeve circumference, garment circumference around waist, etc.).
- Choices exchange. Could shoppers anonymously exchange their choices with their body size information? Or could shoppers rely too much on other selections and worsen their correct choice?
- Automatically rank other shoppers' choices based on similar body sizes. Automatic suggestions for the right fit based on information from the clothing manufacturer.

Homework assignment

- Project problem definition (individual).
 - Each group member writes half an A4 page
 - Everyone, even within the same group, must find his/her own problem (not the same one).
-
- Deadline for submission is October 7, 2021 at 23:59.

Problem definition content

- Project name
- Concise problem description
- Project goals,
- Initial ideas for solving the problem

Literatura

- [1] „Study Guide to Accompany Shari Lawrence Pfleeger's Software Engineering: Theory and Practice” By Forrest Shull and Roseanne Tesoriero
- [2] 2012-2013 3. Letnik RIN Tutorials

Feasibility

Answers the questions:

- It should answer the three key questions:
- **Three key questions:**
 - **Is the project technically feasible? (Can we build it?)**
 - **Is the project economically feasible? (Will it provide business value?)**
 - **Is the project operationally feasible? (Can the product satisfy all the needs from environment needs (are developers available, will it satisfy all the laws and agreements, can it be used...))**
- Additional questions:
 - Which are advantages provided by the project?
 - What is the extent of the project?

The purpose

- Feasibility study enables decision to start the project or not.

Technical feasibility:

- For each alternative implementation technical details have to be considered for all the following phases (including maintenance) and estimate the risks.
- Technical personnel must be included (those that may later become a part of a project team).

Digital Music Download Project Executive Summary

Carly Edwards and Jason Wells created the following feasibility analysis for the Tune Source Digital Music Download Project. The System Request is attached, along with the detailed feasibility study. The highlights of the feasibility analysis are as follows:

Technical Feasibility

The Digital Music Download system is feasible technically, although there is some risk.

Tune Source's risk regarding familiarity with music download applications is moderately high.

- The Marketing Department has little experience with a subscription-based business model.
- The IT department has strong knowledge of the company's existing Web-based CD sales system, but it has not worked with music downloads or customer subscriptions.
- Numerous music download sites exist on the Internet.

Tune Source's risk regarding familiarity with the technology is moderately low.

- The IT department has knowledge of the current Web-based order entry system and the databases and Internet technology it uses.
- The IT department has no direct knowledge of the technology required to store and deliver digital music downloads; however, many of the technical issues will be the responsibility of the ISP.
- Consultants are readily available to provide help in this area.

The project size is considered medium risk.

- The project team will likely consist of 10 or fewer people.
- Business user involvement will be required.
- The project time frame is somewhat critical, since the system is needed to maintain our competitive position in the market.

The compatibility with Tune Source's existing technical infrastructure should be good.

- An Internet infrastructure is already in place at the retail stores and corporate headquarters.
- The ISP should be able to scale its services to accommodate the new Digital Music Download system.

Economic feasibility:

- Costs:
 - One time costs of equipment (HW+SW), education, SW, consultations, support...
 - Variable costs, salaries, assets, maintenance, rents...
- Benefits:
 - Savings (salary savings, material, increase of production, reduced maintenance and usage costs...)
 - Contributions (better customer services, improved management capabilities, control of production, reduction of failures and errors...)

How to estimate project costs?

- By splitting the system into components. Easier to do estimation for smaller components.
- Using historical data (previous projects)
- Estimation of personnel costs (development and maintenance). A time plan of cost changes shall be made.
- Using organizational standards to estimate additional costs (management, administration, rooms, electricity...)

- The result of the feasibility study is a feasibility report, which is given to the management and to users. They have to estimate:
 - Are alternatives acceptable?
 - Are solving the right problem?
 - Whether any of alternatives offers advantages over the others or assures project success? Consideration of the expected SW lifespan is needed, typically 5-7 years.
- Users and management select one of the alternatives if they decide to start the project.
- Many projects never start and ‘die’ earlier -at this point.

Example of economic feasibility [1]

- In this example, benefits accrue because the project is expected to increase sales, reduce customer complaint calls, and lower inventory costs.
- For simplicity, all development costs are assumed to occur in the current year 2012, and all benefits and operational costs are assumed to begin when the system is implemented at the start of 2013, and continue through 2016.

	2012	2013	2014	2015	2016	Total
Benefits						
Increased sales		500,000	530,000	561,800	595,508	2,187,308
Reduction in customer complaint calls ^a		70,000	70,000	70,000	70,000	280,000
Reduced inventory costs		68,000	68,000	68,000	68,000	272,000
Total Benefits^b	638,000	668,000	699,800	733,508	2,739,308	
Development Costs						
2 servers @ \$125,000	250,000	0	0	0	0	250,000
Printer	100,000	0	0	0	0	100,000
Software licenses	34,825	0	0	0	0	34,825
Server software	10,945	0	0	0	0	10,945
Development labor	1,236,525	0	0	0	0	1,236,525
Total Development Costs	1,632,295	0	0	0	0	1,632,295
Operational Costs						
Hardware		50,000	50,000	50,000	50,000	200,000
Software		20,000	20,000	20,000	20,000	80,000
Operational labor		115,000	119,600	124,384	129,359	488,343
Total Operational Costs	185,000	189,600	194,384	199,359	768,343	
Total Costs	1,632,295	185,000	189,600	194,384	199,359	2,400,638
Total Benefits – Total Costs	(1,632,295)	453,000	478,400	505,416	534,149	338,670
Cumulative Net Cash Flow	(1,632,295)	(1,179,295)	(700,895)	(195,479)	338,670	
Return on Investment (ROI)	14.1%	(338,670/2,400,638)				
Break-even Point	3.37 years	[3 years of negative cumulative cash flow + (534,149 – 338,670)/534,149 = .37]				

Operational feasibility

- Is this project supported by:
 - Senior management
 - Users
 - Other stakeholder
 - Is the project strategically aligned with the business?
 - Will the project be successfully accepted by the customer?

Requirements analysis

- This phase happens after the project starts - after the feasibility study
- The aim of requirements analysis and definition is to describe the system from user perspective in details:
 - Functional requirements: use cases for all interactions of user with the system.
 - Other, nonfunctional, requirements, that limit design or implementation:
 - Operational requirements (security, safety, usefulness, performance...)
 - Evolution requirements (testing ability, maintenance ability, extensibility, upgradability...)
- Requirements analysis is performed by system analyst.
- Incorrect, incomplete or ambiguous specification often leads to project failure or disputes.

Requirements analysis

- Challenges:
 - Users often do not know well or cannot describe their needs, or they do not know how SW could help.
 - User wishes change with time.
 - Users have inconsistent requirements.
 - Users have difficulties estimating what is feasible (also related to costs)
 - Users do not distinguish between wishes and requirements
 - System analyst has only a limited knowledge of the system's field of use.
 - Customer and user may not be the same person.

Requirements analysis

- The process:
 - Obtain knowledge from the field of use.
 - Getting acquainted with the problem (interviews, questionnaires, searching of interdependences, identifying gaps).
 - Study of existing systems.
 - The analysis often from outside in – from required outputs to required inputs, processes, storage needs.
 - System description (describing a system textually, with diagrams...)
 - Revision of the requirements analysis, together with users.
 - This process may continue with another iteration of all mentioned here.

Requirements analysis

- The result is a document called Software requirements specifications (SRS), (sometimes incorrectly called functional specifications -FS)
 - Informs users of the system such that they can estimate if the proposed system is correctly addressing their requirements, if the developer understanding of a problem is correct.
 - Splits the problem into smaller pieces – components.
 - The document is later used for system design.
 - The document is later used for system validation.

Requirements analysis

- Document contents must be adjusted to fit the project needs and **typically include:**
 - Introduction (purpose, extent, definitions, references to other documents)
 - General description (relations to other systems, function and component overview, user specifics, general limitations)
 - Description of requirements for each of the components (description, inputs, processing, outputs)
 - External interface requirements (formats, Hw, connections/communication to other systems)
 - Required performance
 - Design limitations (standards, HW limitations...)
 - Other requirements

Requirements analysis

- The requirements analysis phase ends with review:
 - Review of all requirements:
 - Validity. Do system functions comply with user needs?
 - Consistency. Are there requirements that oppose each other?
 - Completeness. Are all the functions needed by users considered?
 - Reality. Can requirements be satisfied with the proposed technology and with the cost limitations?
 - Review of individual requirements:
 - Testability. Can the fulfillment of the requirement be checked?
 - Understandability. Is the requirement correctly understood?
 - Traceability. Is it clear why certain requirement exist?
 - Adaptability. Can the requirement be changed or modified without high influence on other requirements?

IEEE Standard SRS (Software Requirements Specifications) Template

- **1. Introduction**
 - 1.1. Purpose
 - 1.2. Scope
 - 1.3. Definitions, acronyms & abbreviations
 - 1.4. References
 - 1.5. Overview
- **2. Overall description**
 - 2.1. Product perspective
 - 2.1.1. System interfaces
 - 2.1.2. User interfaces
 - 2.1.3. Hardware interfaces
 - 2.1.4. Software interfaces
 - 2.1.5. Communications interfaces
 - 2.1.6. Memory constraints
 - 2.1.7. Operations
 - 2.1.8. Site adaptation requirements
 - 2.2. Product functions
 - 2.3. User characteristics
 - 2.4. Constraints
 - 2.5. Assumptions and dependencies
 - 2.6. Apportioning of requirements

- **3. Specific Requirements**
 - 3.1 External interface requirements
 - 3.1.1 User interfaces
 - 3.1.2 Hardware interfaces
 - 3.1.3 Software interfaces
 - 3.1.4 Communication interfaces
 - 3.2 Specific requirements
 - 3.2.1 Sequence diagrams
 - 3.2.2 Classes for classification of specific requirements
 - 3.3 Performance requirements
 - 3.4 Design constraints
 - 3.5 Software system attributes
 - 3.5.1 Reliability
 - 3.5.2 Availability
 - 3.5.3 Security
 - 3.5.4 Maintainability
 - 3.6 Other requirements
 - **4. Supporting information**
 - 4.1 Table of contents and index
 - 4.2 Appendixes
- Note: History of versions of this document with author/contributor info may be included before the main sections of the document.**

Homework assignment

- Feasibility study for solving your problem, existing alternatives (individual)
 - Technical feasibility
 - Economics feasibility
 - Opeartional feasibility

Literature

- [1] Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. *Systems analysis and design*. John wiley & sons, 2008.

Modeling and diagramming (UML and others)

Why diagrams?

- Clear description of models.
- The basic tool for modeling:
 - Enable description of complex systems
 - Unambiguous presentation contributes to easier understanding of a problem and a solution.
 - Make modeling of system structure and behavior easier.

Diagrams and models

- Diagram is a **partial** graphic presentation of a system model.
- Model can include multiple diagrams and documentation.
 - Example: the use-case model consists of a use-case diagram and a document describing the use-case.
- This lecture is limited only to diagramming techniques (diagrams) and NOT the overall models.

Diagrams for SW product development

- **Flowchart**
- **Data Flow Diagram, DFD**
- **Entity Relationship Diagram, ER diagram**
- **UML diagrams** (Unified Modeling Language) – diagramming technique for various diagram types, specified for SW engineering.
- **Others** (depending on the needs)

Diagram (and model) types

- **Context**
 - Description of a system from outside – in his environment.
- **Behavioral**
 - System behavior: use-cases, process behavior, data flow, state transitions...
- **Structural**
 - Structure of a system, components, objects, data...

Diagramming (model) domains

- **Application domain** For requirement analysis
 - System environment
- **Solution domain** System design, component/object design
 - System development technologies

Diagramming recommendations

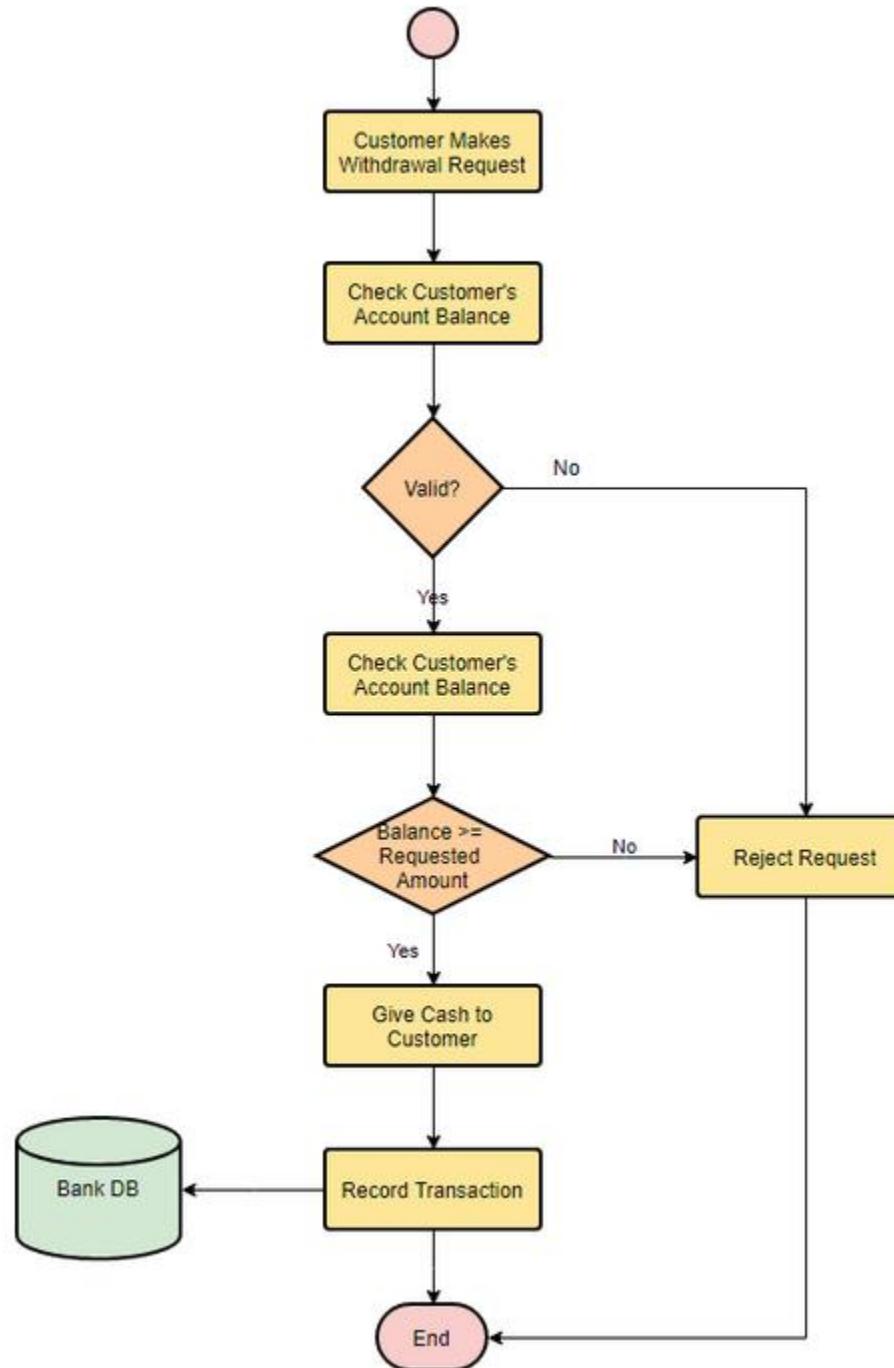
- Avoid line crossing.
- In the case of line crossing use jump instead.
- Avoid diagonal and curved lines.
- Elements shall have comparable size.
- Show only what is needed.
- Use renowned notations whenever possible.
- Reorganize large diagrams in multiple smaller ones.
- Focus on the contents first, then on the appearance.
- Organize diagrams from left to right and from top to bottom.
- Use suitable names (considering the meaning and usage area)
- Mark unknowns with comments and question marks.
- Color usage.

Flowchart diagram

Flowchart definition [1]

- A flowchart is a picture of the separate steps of a process in sequential order that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. [1]
- Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields for illustrating a solution model to a given problem. [1]

[1]



Flowchart

- Diagram consists of symbols for start, end, control flow, process steps, I/O operations, decisions...

Flowchart symbols

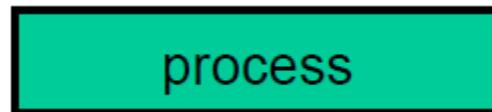
- Start, end: rounded rectangles or circles



- Control flow: arrows



- Process steps: rectangles

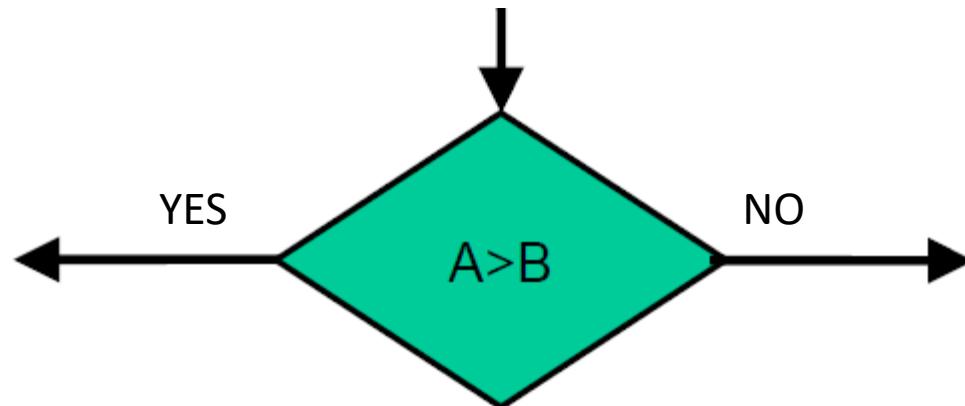


Flowchart symbols (2)

- I/O operations: parallelograms



- Conditions, decisions: rhomboids, labels at exiting arrows.



When to Draw Flowchart Diagram? [1]

- Using a flowchart has a variety of benefits:
 - It helps to clarify complex processes.
 - It identifies steps that do not add value to the internal or external customer, including: delays; needless storage and transportation; unnecessary work, duplication, and added expense; breakdowns in communication.
 - It helps team members gain a shared understanding of the process and use this knowledge to collect data, identify problems, focus discussions, and identify resources.
 - It serves as a basis for designing new processes.

How to draw a Flowchart Diagram? [1]

- How do you get from a complex task to an organized flowchart describing how to do it?
- Start with a flowchart containing just the task.
- Now break it down into smaller, more specific steps in another flowchart.
- Then, consider any possible exception in the flow, if so, add decision node for the alternative paths.
- Keep on repeating this process until you've reached steps that are simple enough for everyone to fully understand it.

Literature

- [1] <https://online.visual-paradigm.com/diagrams/tutorials/flowchart-tutorial/>

DFD, Data Flow Diagram

DFD

- Main characteristics:
 - Modeling of data flow, not control flow.
 - Logical view on a system, not physical.
 - Convenient for communication with users, management and information system experts.
 - Suitable for analysis of existing and proposed systems.
 - Relatively simple technique.

DFD [1]

- A neat and clear DFD can depict a good amount of the system requirements graphically.
- It shows how information enters and leaves the system, what changes the information and where information is stored.
- The purpose of a DFD is to show the scope and boundaries of a system as a whole.
- It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

DFD

- Show how data is processed in the sense of inputs, processes and outputs

DFD symbols (1) [1]

- **External Entity**

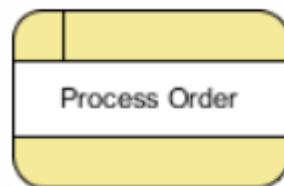
- An external entity can represent a human, system or subsystem.
- It is where certain data comes from or goes to.
- It is external to the system we study, in terms of the business process.
- For this reason, people used to draw external entities on the edge of a diagram.



DFD symbols (2) [1]

- **Process**

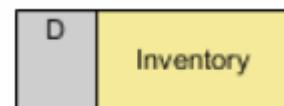
- A process is a business activity or function where the manipulation and transformation of data take place.
- A process can be decomposed to a finer level of details, for representing how data is being processed within the process.



DFD symbols (3) [1]

- **Data Store**

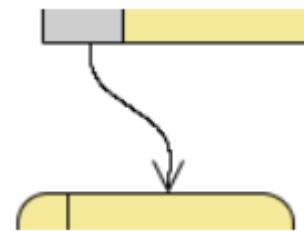
- A data store represents the storage of persistent data required and/or produced by the process.
- Here are some examples of data stores: membership forms, database tables, etc.



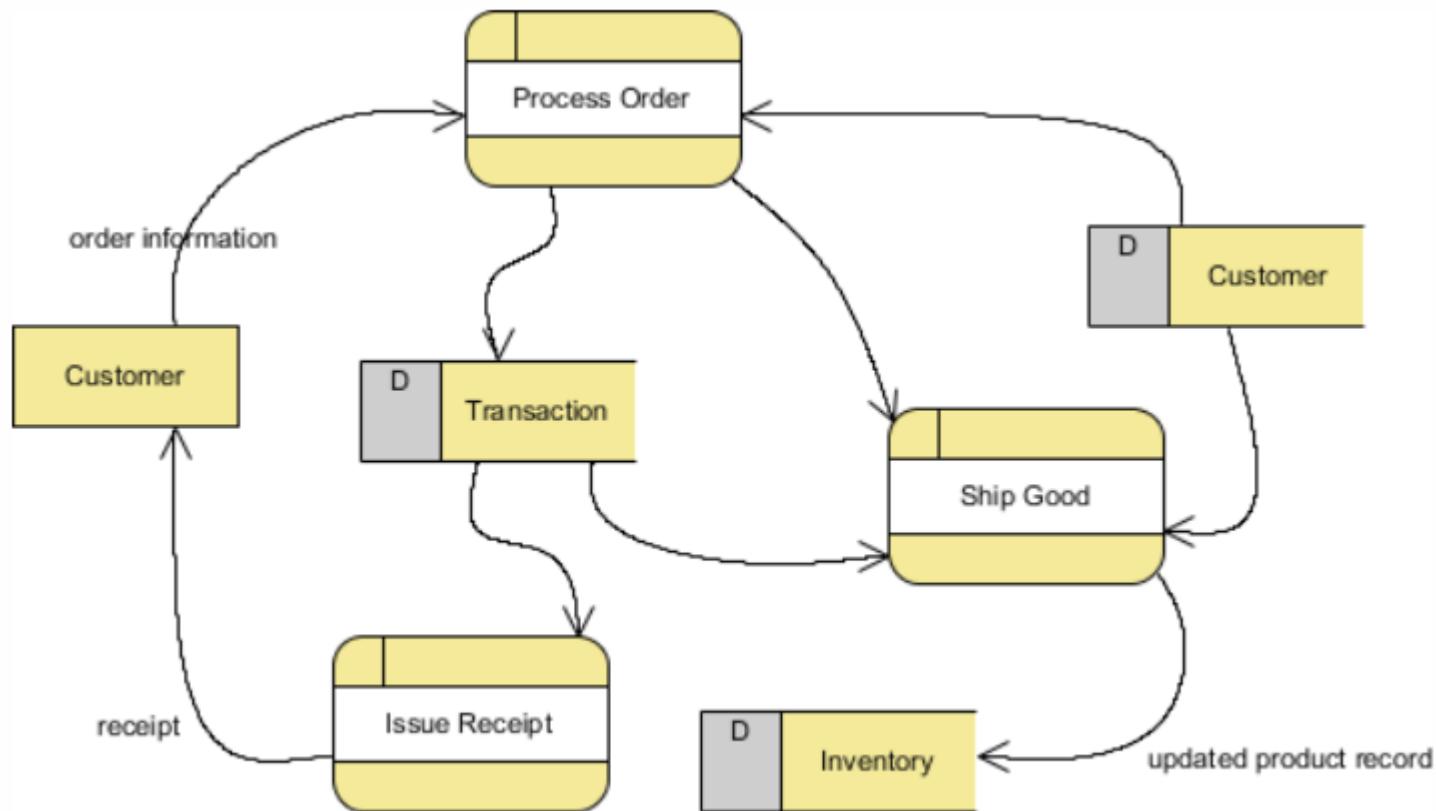
DFD symbols (4) [1]

- **Data Flow**

- A data flow represents the flow of information, with its direction represented by an arrowhead that shows at the end(s) of flow connector.



Example DFD [1]



Literature

- [1] <https://www.visual-paradigm.com/tutorials/data-flow-diagram-dfd.jsp>

ERD, Entity Relationship Diagram

ERD definition [1]

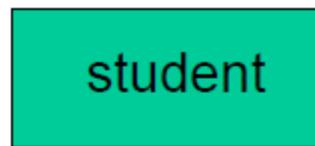
- Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design.
- An ERD contains different symbols and connectors that visualize two important information: **The major entities within the system scope, and the inter-relationships among these entities.**
- And that's why it's called "Entity" "Relationship" diagram (ERD)!

ERD

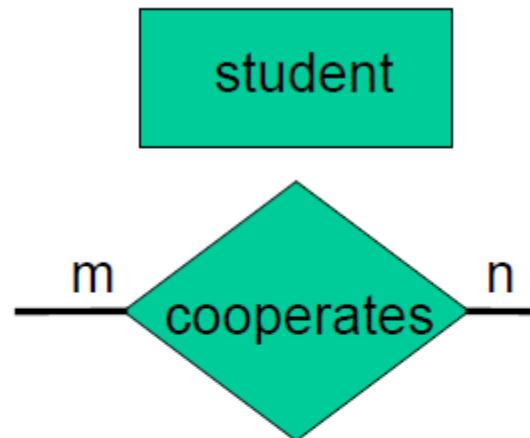
- Intended for illustration of a data structure.
- Definitions:
 - **Entity** – a real world object, which can be distinguished from other objects.
Entity consists of a set of attributes.
 - **Entity set** – a set of similar entities, with the same set of attributes.
 - **Attribute** – a characteristic of an entity.
 - **Relationship** – relation between two or more entities
 - **Key** – an attribute that enables differentiation between entities.

ERD basic symbols

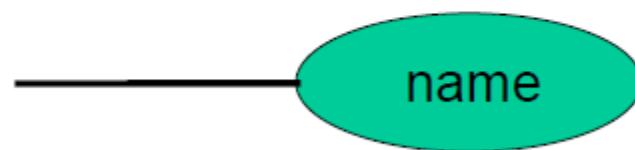
- entity (entity set)



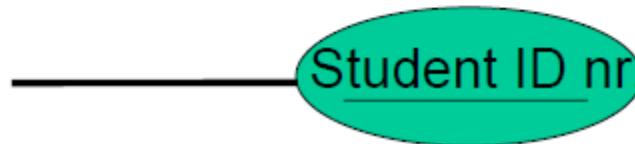
- relation



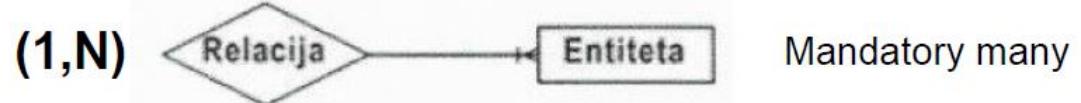
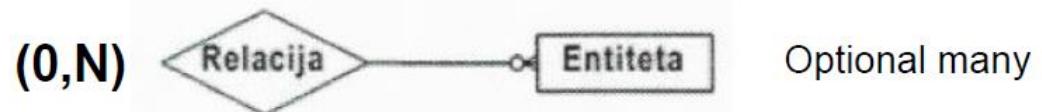
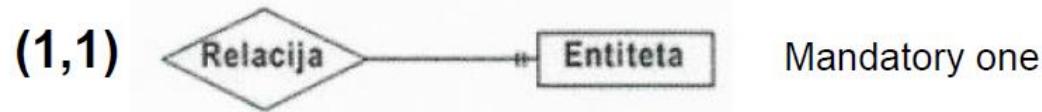
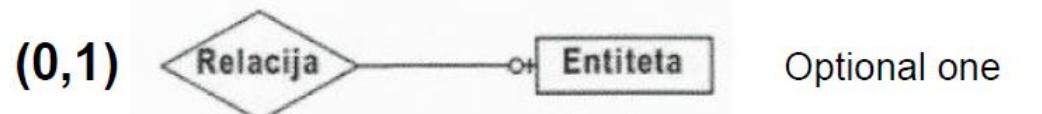
- attribute



- key

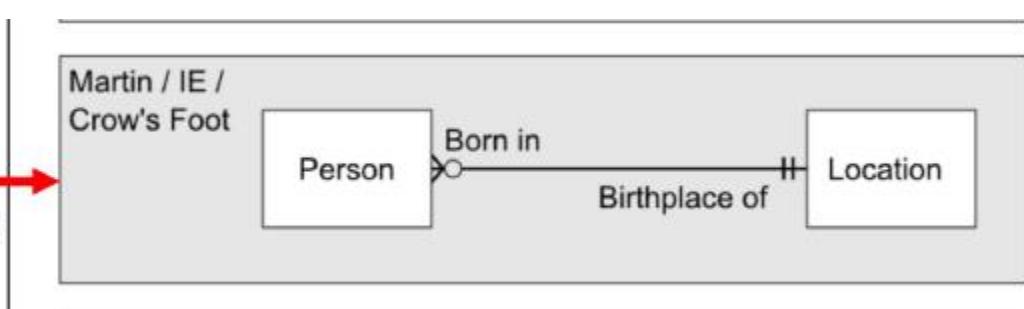


Cardinality



Unified methodology for the development of information systems

EMRIS



Tools

- SQL power architect
 - <http://www.sqlpower.ca/page/architect>
- Mysql Workbench
 - <https://www.mysql.com/products/workbench/>
- Toad data modeler
 - <http://www.quest.com/Toad-Data-Modeler/>
- Others...

Literature

- [1] <https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/>

Unified Modeling Language (UML)

UML introduction

- UML - standardized notation with proposed diagrams for building a software system model
- System model is a complete description of the system viewed from a particular perspective
- UML diagrams can be subdivided into two groups:
 - Structure diagrams
 - **Class diagram**
 - **Component diagram**
 - **Deployment diagram**
 - **Object diagram**
 - Package diagram
 - Behavior diagrams
 - **Activity diagram**
 - **State machine diagram**
 - **Use case diagram**
 - Communication diagram
 - Interaction overview diagram
 - **Sequence diagram**
 - Timing diagram

Behavior diagrams

- **Use case diagram** – Shows the system functionality: the system's high-level requirements.
- **Activity diagram** - Depicts the control flow: flow of different activities and actions
- **Sequence diagram** - describes the sequence of messages and interactions that happen between actors and objects.
- **State machine diagram** - is used to describe the different states of a component within a system.

Structure diagrams

- **Component diagram** - breaks down the system into smaller components to depict the system architecture.
- **Deployment diagram** - is used to visualize the relation between software and hardware.
- **Class diagram** - depicts classes, alongside with their attributes and their behaviors .
- **Object diagram** - depicts instances (objects) of the classes present at certain time.

Tools

- For the design of UML diagrams, the visual paradigm is used in all the examples below.
 - <https://www.visual-paradigm.com/>
- Download link:
 - <https://www.visual-paradigm.com/download/community.jsp>
- Other tools:
- Eclipse Papyrus ™ Modeling environment
 - <https://www.eclipse.org/papyrus/>
- Bouml
 - <https://www.bouml.fr/>
- There are many other tools as well.

Revision control

Revision control [1]

- In the software development process, **revision control**, also known as **version control** or **source control**, is the management of changes made over time.
- These changes can be to source code, project assets, or any other information that goes into the finished output.
- It permits many people to work on the same parts of a project without worrying that their changes will overwrite the work of anyone else.

Revision control [1]

- The collection of revisions and their metadata is called a **repository** or **repo**.
- The repository represents a step-by-step chronological record of every change made to help project managers revert all or part of the project to a previous state if necessary.

How revisions are made? [1]

- Revision control systems are usually hosted on a networked **server**.
- After the repository is set up, using it generally involves the following steps:
 - 1. If the developer has created a new file that should become part of the project, the file must be **added** to the repository. The file is uploaded to the repository, and anyone else working on the project can see and use the file.
 - 2. If the developer wants to edit a file that is already part of the project, the file must be **checked out**. The act of checking out downloads the desired revision of the file to the developer's **local version** of the project. Usually, the revision that a developer wants to edit is the most recent revision: this revision is known as the „head”.

How revisions are made? (2)[1]

- 3. After the developer edits the file locally and is ready to add it to the official version of the project, the file can be **checked in**. This action is also known as making a **commit**. The developer is asked to write a summary of what changes were made and why. These comments, with the updated version of the file, are uploaded to the repository.
- 4. If someone else has checked in revisions to the same file since the last time the developer checked it out, the system announces that there are conflicts. It calculates the differences line-by-line, and the developers who made the changes must agree upon how their individual changes should be **merged**. The merging is usually done manually: the developers compare the conflicting versions and decide how to resolve them into one document.

How revisions are made? (3)[1]

- 5. If there are no conflicts, the new version is updated in the repository, and the entire project receives a new **revision number**, permanently and uniquely identifying its current state.

Branching the development tree [1]

- Experimental changes are often made to the main version of a software project.
- Using revision control, these changes can be made to a separate copy of the project without interfering with development of the main version.
- The terminology for this approach uses the metaphor of a tree: the main version of the project is called the **trunk**, and experimental versions are known as **branches**.

Centralized vs. distributed systems [1]

- If a revision control systems uses a centralized repository, its data is contained in a single database containing the authoritative version of all project files.
- Other systems employ a distributed model. In these systems, changes can be checked, and then synchronized between repositories.

List of revision control software (1) [1]

- Autodesk Vault
 - A specialized revision control system for various Autodesk CAD software products including Autodesk Inventor Professional, AutoCAD Mechanical, AutoCAD Electrical, and Autodesk Revit. Allows designers to re-use previous designs rather than starting from scratch, and to access complicated 3D schematics from any point in their design process.
- Azure DevOps Server
 - A software and system package developed by Microsoft, providing source code management and collaborative tools for software development. Azure DevOps Server also provides applications that facilitate project reporting, automated builds, lab management, testing, and release management. Designed to integrate with most IDE or editor tools.

List of revision control software (2) [1]

- Bazaar
- A distributed revision control system also known as **GNU Bazaar**. Its development is sponsored by Canonical, who also develops the Ubuntu Linux distribution. Runs on Linux, OS X, and Windows. Facilitates working with repositories from other systems such as CVS, Subversion, Git, Mercurial, Darcs, and Perforce.
- Git
- A distributed revision control system designed and implemented by **Linus Torvalds** for use in the development of the Linux kernel. First released in 2005, Git is now the most widely-used version control system in the world. The software is free, open source, and released under the GPL.

Homework assignment - Team work

- Groups propose/select their topic and find an agreement on the general solution direction.

One member of the group must upload their group's choice (a pdf with: project problem definition, feasibility study).

Also, For your homework please open a project or group on GitHub and in Google Docs (one GitHub project and one Google Docs document per group).

Put the link (share it with us) for the GitHub and Google Docs documents in the uploaded pdf file in the e-Classroom.

Literature

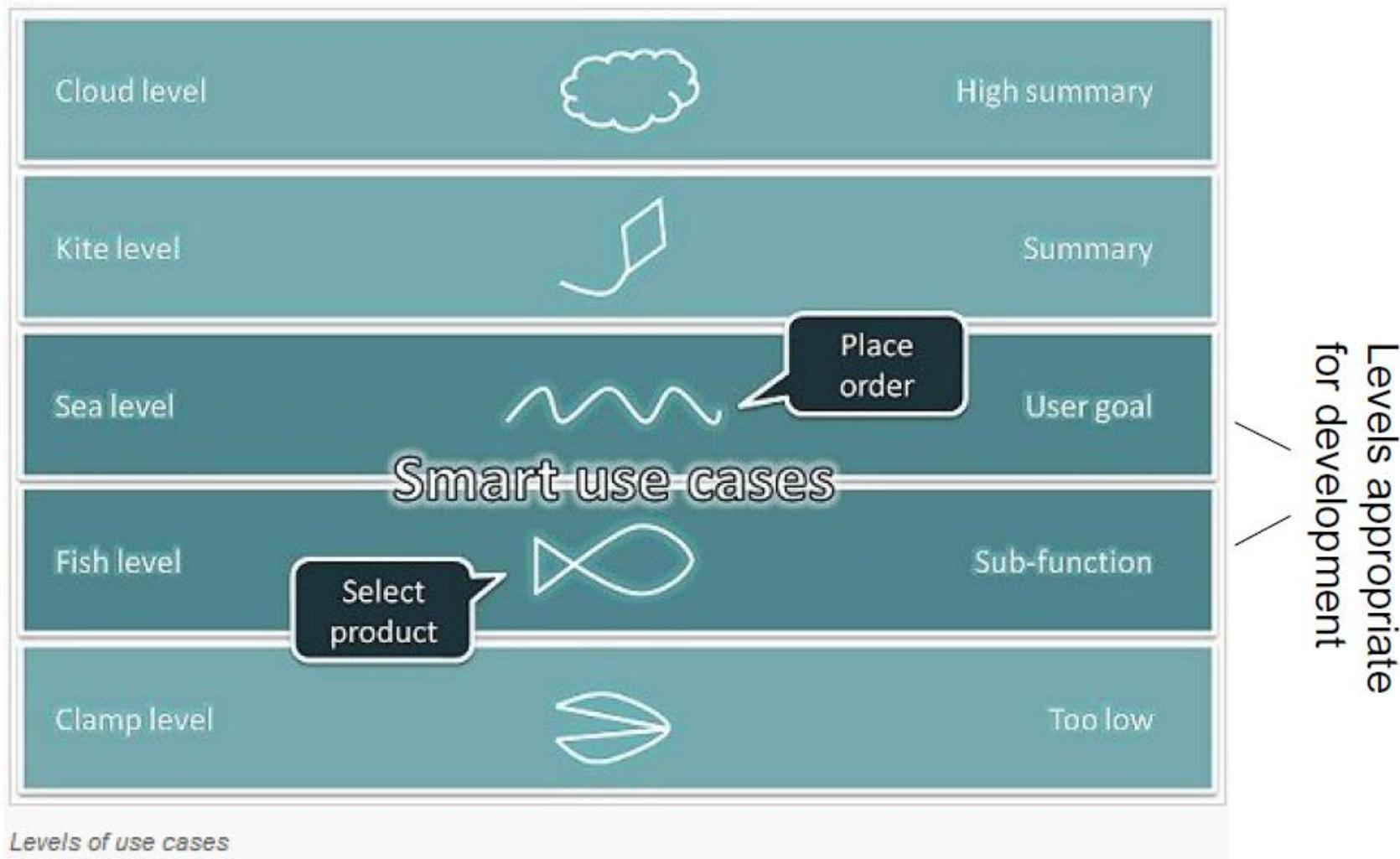
- [1] <https://www.computerhope.com/jargon/r/revision-control.htm>

UML use case diagram

UML use case diagram - definition

- **Use case diagram** – Shows the system functionality: the system's high-level requirements

Use case diagram: levels



- <https://www.youtube.com/watch?v=FNkuGEr1gB4>

Actor

- An actor can generally be an individual user of a system, device, database or other system that communicates with the described system, exchanges data and obtains results from it.
- Questions for determining the actors:
 - Who is interested about a particular requirement that a system must meet?
 - Who in the organization will use the system?
 - Who will gain something by using the system?
 - Who will supply the system with the data, who will use it and who will remove it?
 - Who will support and maintain the system?
 - Does the system use external resources?
 - Does one person play several different roles?
 - Do more people play the same role?

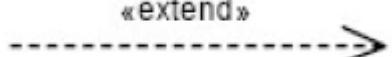
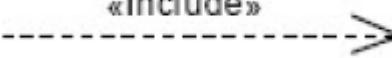


Use case

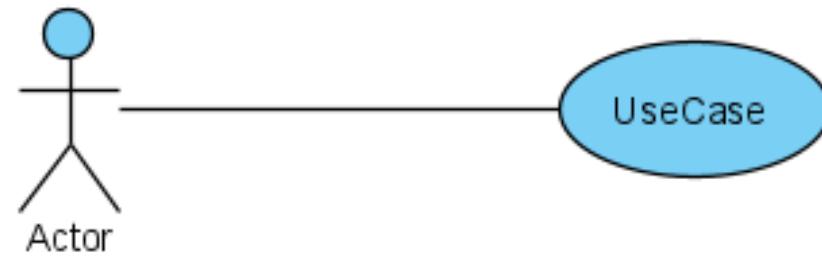
- A use case is a connected unit of functionality provided by a system or subsystem and expressed in message sequences exchanged between the system and one or more users. It ends with a measurable result that has some significance for a particular actor.
- The following questions can be used to determine the use cases of the system:
 - What are the tasks of each actor?
 - Will any of the actors enter, store, delete or view information in the system?
 - Which use case will enter, store, delete or review this information?
 - Will any of the actors have to notify the system of sudden external changes?
 - Whether any of the actors must be informed of certain events in the system
 - Which use case will support and maintain the system?
 - Do the use cases support all the functional requirements of the system?



Types of relations between use cases

Relations	Description	Notation
association	Communication link between the actor and the use case in which he participates.	
extend	The use case at the beginning of the arrow extend the behavior of the use case at the end of the arrow.	
include	The use case at the beginning of the arrow contains the behavior of the use case at the end of the arrow.	
generalization	The relationship between a general and a more specific use case, which inherits the characteristics of the general use case and adds new ones.	

The relationship between the actor and the use case



- Note: Use cases connected to actors are those that are the reason of actor using the system - so a just correctly high level ones, or better said "sea level" ones.

Example 1 [2]

- Task

Model the hotel management system given in the following description using the UML use case diagram.

Hotel guests can browse the hotel offer online and make a room reservation. When booking a room, guests can optionally reserve a parking space and purchase some of the following options: Breakfast, Half Board or Full Board. The hotel reception can rent out a room and charge for it. When renting a room, the receptionist must verify the reservation and program the key (card) to open the room. When billing, the receptionist issues an invoice. The system administrator manages the available rooms, enters parking spaces, and processes the hotel price list. Managing available rooms includes adding new rooms, deleting existing rooms, and entering room information (number of beds, additional amenities, etc.).

Question

- How many actors we have ?



Guests



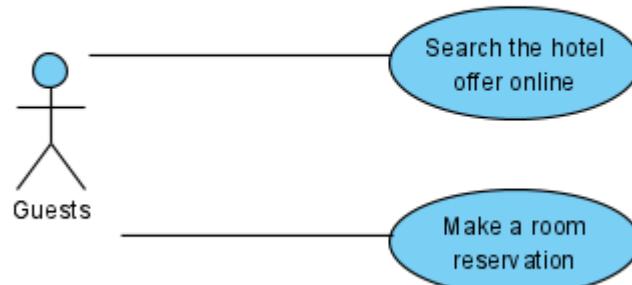
The system administrator



Receptionist

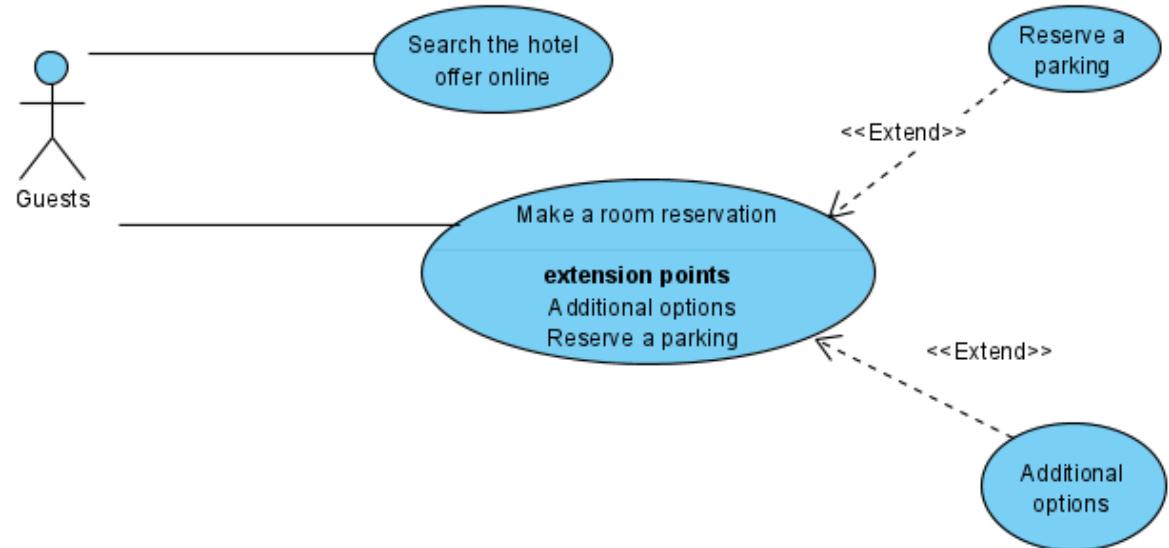
Guests – use cases (1)

- Hotel guests can browse the hotel offer online and make a room reservation.



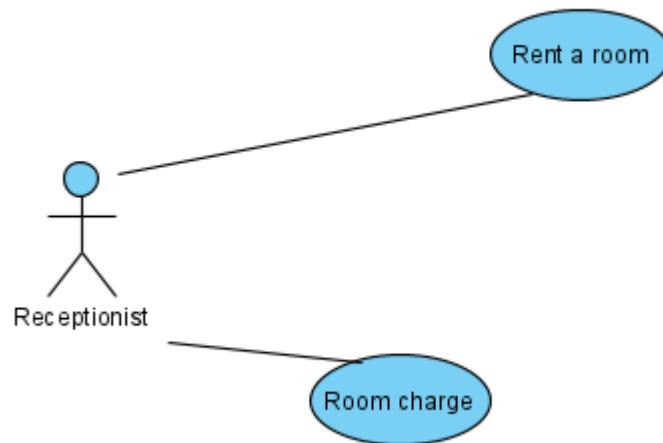
Guests – use cases (2)

- When booking a room, guests can optionally reserve a parking space and purchase some of the following options: Breakfast, Half Board or Full Board.
- Note: You do not need to draw this as it is obvious from "extended" use cases.



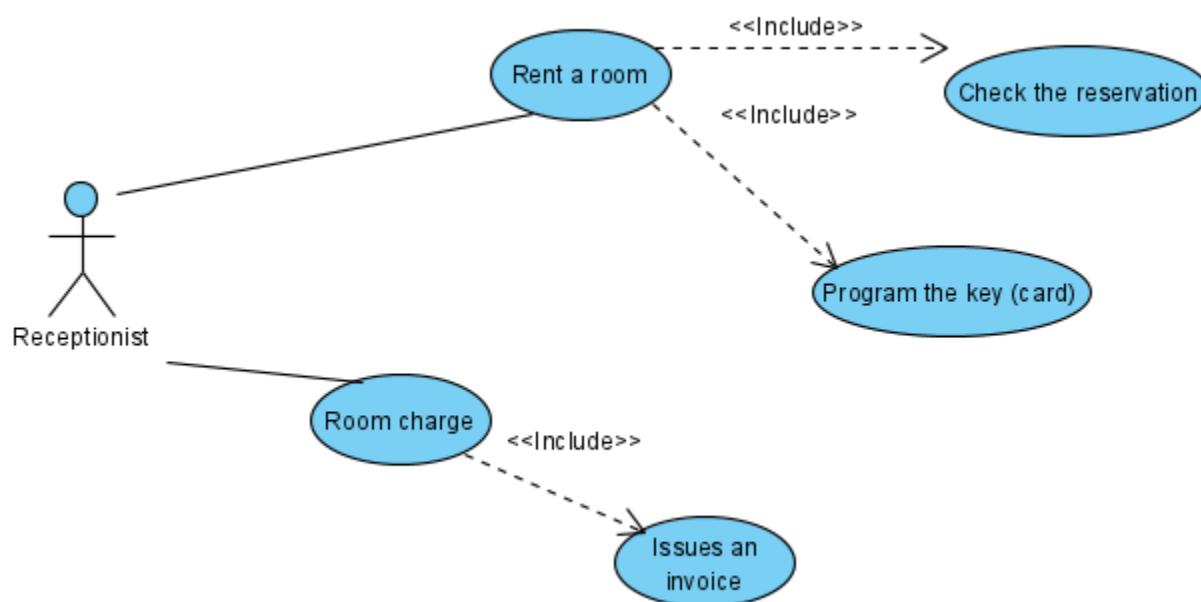
Receptionist – use cases (1)

- The hotel reception can rent out a room and charge for it.



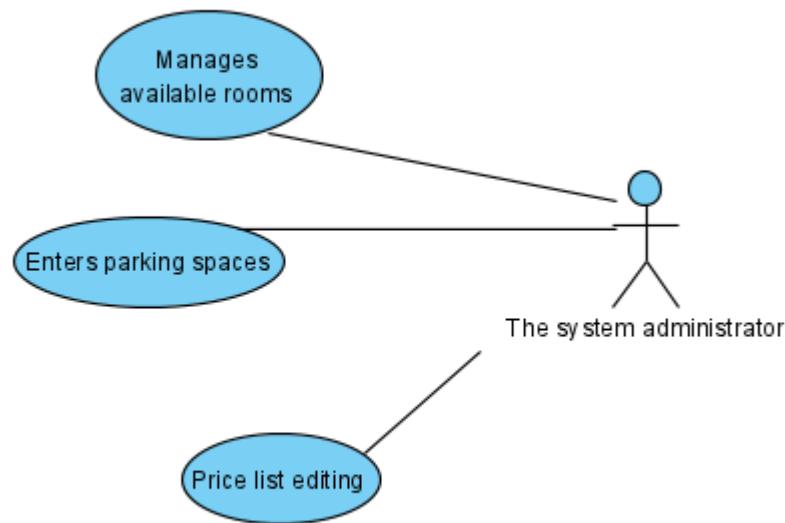
Receptionist – use cases (2)

- When renting a room, the receptionist must verify the reservation and program the key (card) to open the room. When billing, the receptionist issues an invoice.



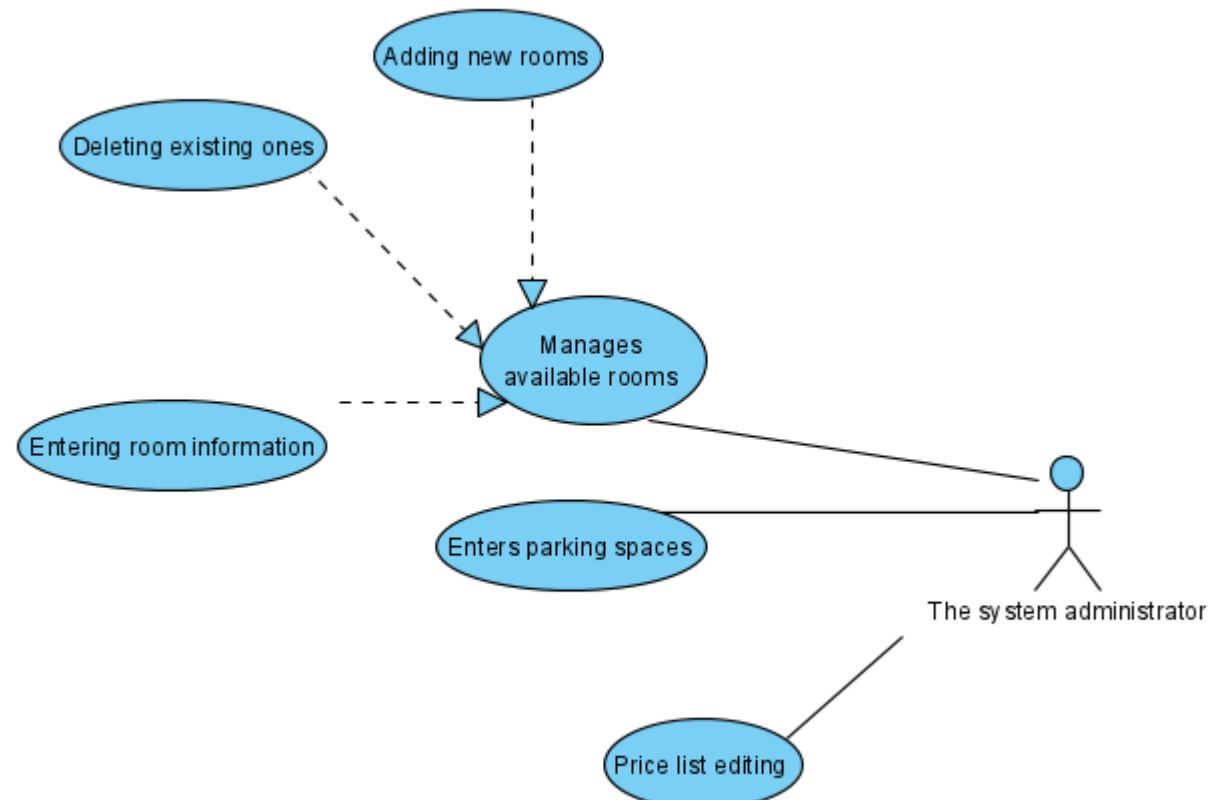
The system administrator – use cases (1)

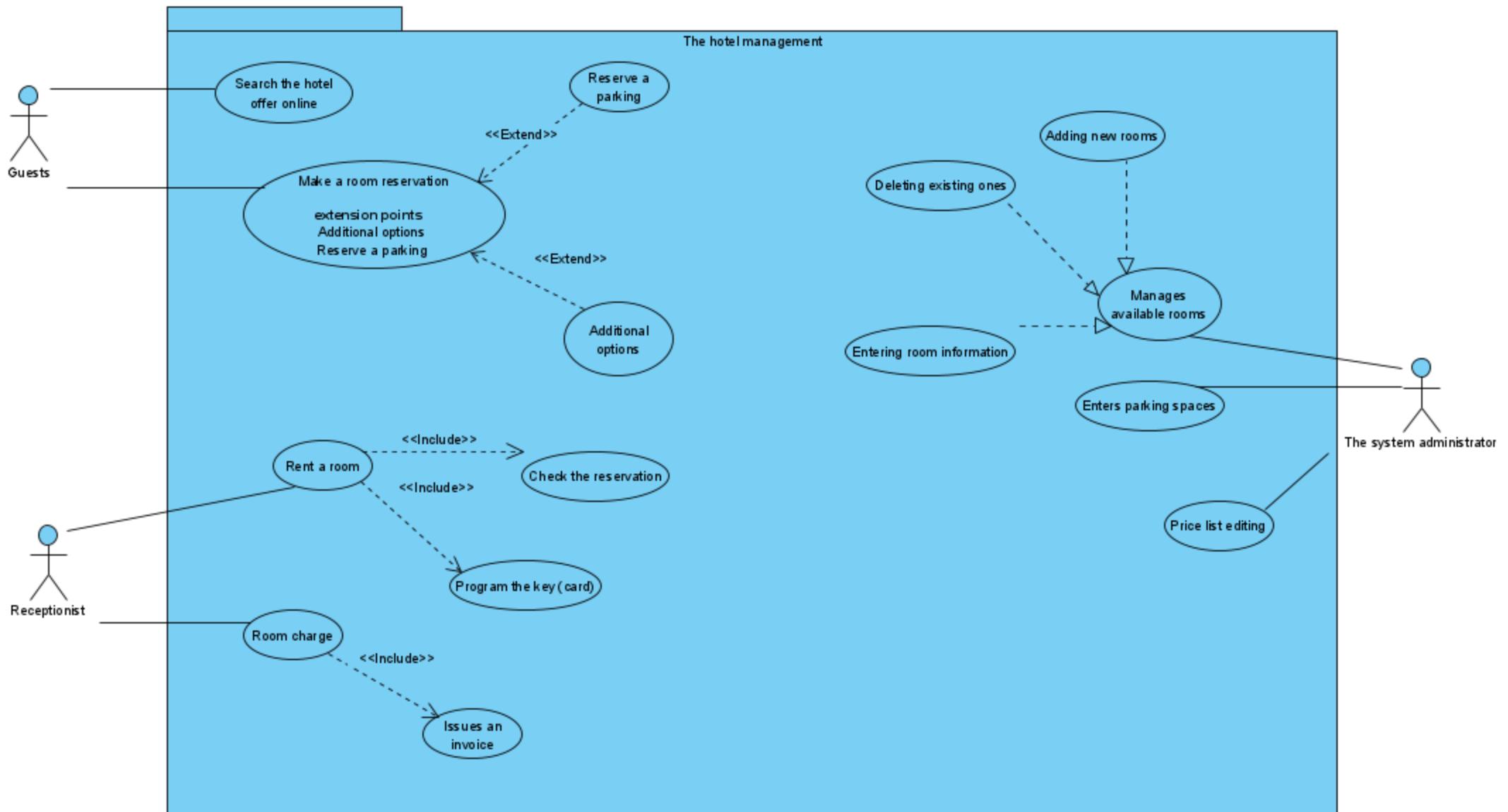
- The system administrator manages the available rooms, enters parking spaces, and processes the hotel price list.



The system administrator – use cases (2)

- Managing available rooms includes adding new rooms, deleting existing rooms, and entering room information (number of beds, additional amenities, etc.).





Comments:

- Reserving a parking space and purchasing options do not necessarily have to be done when booking a room, and therefore these forms are linked to the main form by an "extend" link. In contrast, when renting a room, it is necessary to check the reservation, so the link "include" is selected.
- Additional options for breakfast, half board and full board are combined in one form. This means that when implemented, they should be offered as a list of options from which the user selects (displays) the desired options.

Example 2

- Task

Model the simple online testing system from the following description using the UML use case diagram.

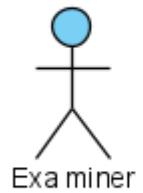
An examiner organizing an online exam must prepare a question bank. In addition, the question bank contains two optional types of questions, one is practice questions and the other is theoretical questions.

The examiner's task is also to prepare the exam. There are two ways to take the exam: the theoretical exam or the practical exam.

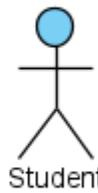
The student can take an exam and review example results. The procedure for taking the exam includes login and registration for the exam.

Question

- How many actors we have ?



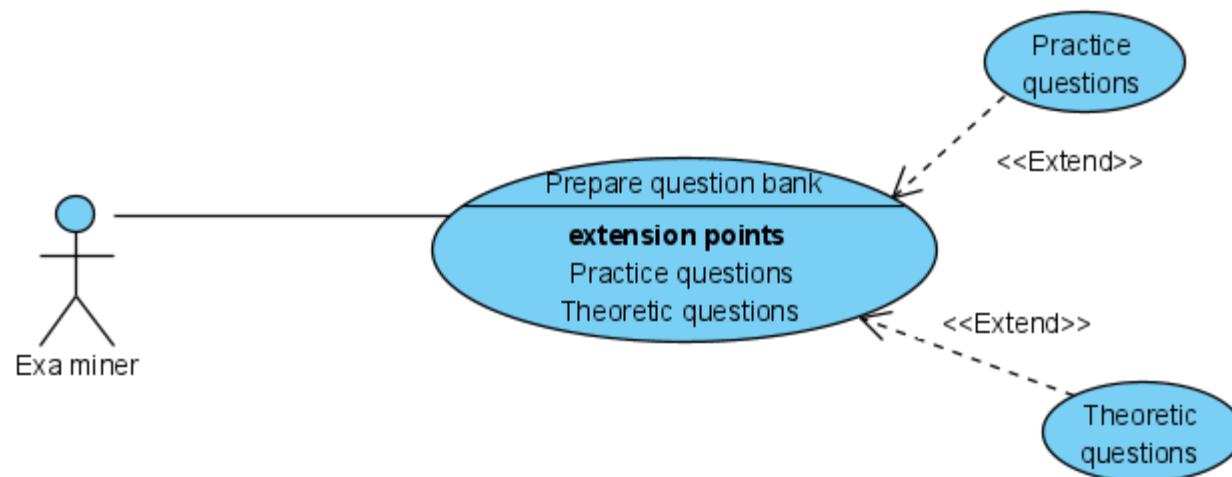
Examiner



Student

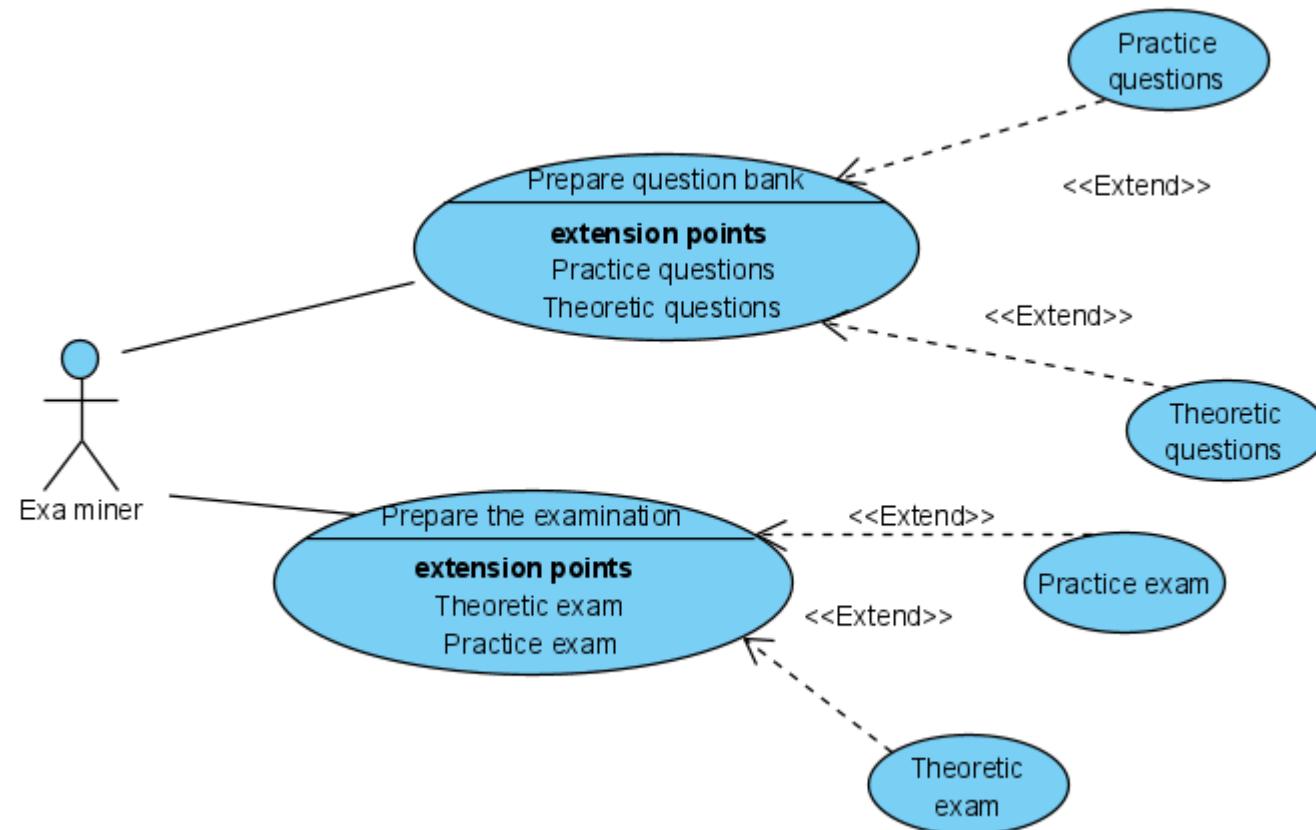
Examiner – use cases (1)

- An examiner organizing an online exam must prepare a question bank. In addition, the question bank contains two optional types of questions, one is practice questions and the other is theoretical questions.



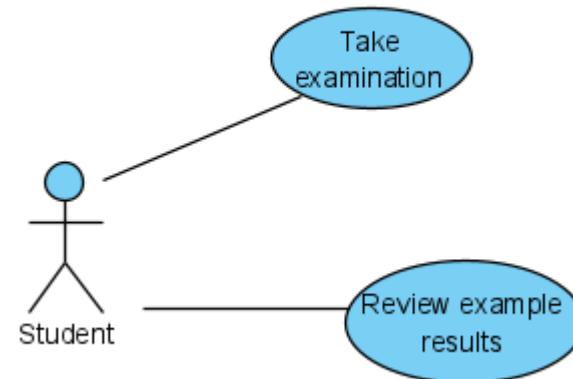
Examiner – use cases (2)

- The examiner's task is also to prepare the exam. There are two ways to take the exam: the theoretical exam or the practical exam.



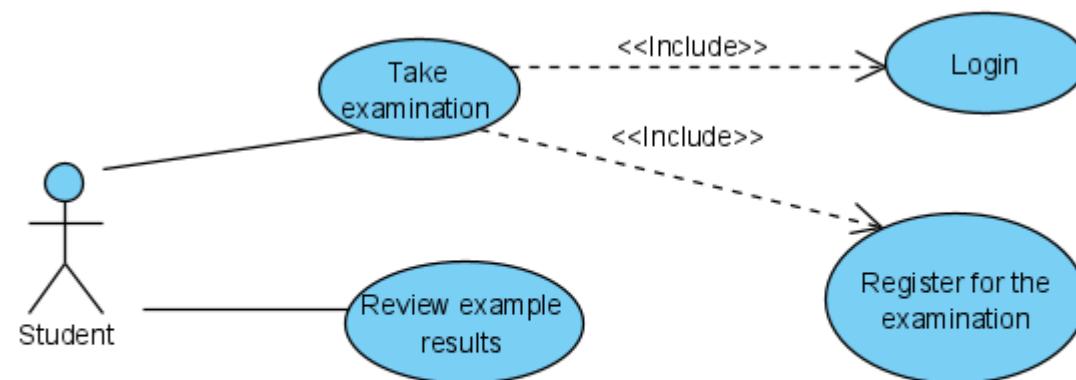
Student – use cases(1)

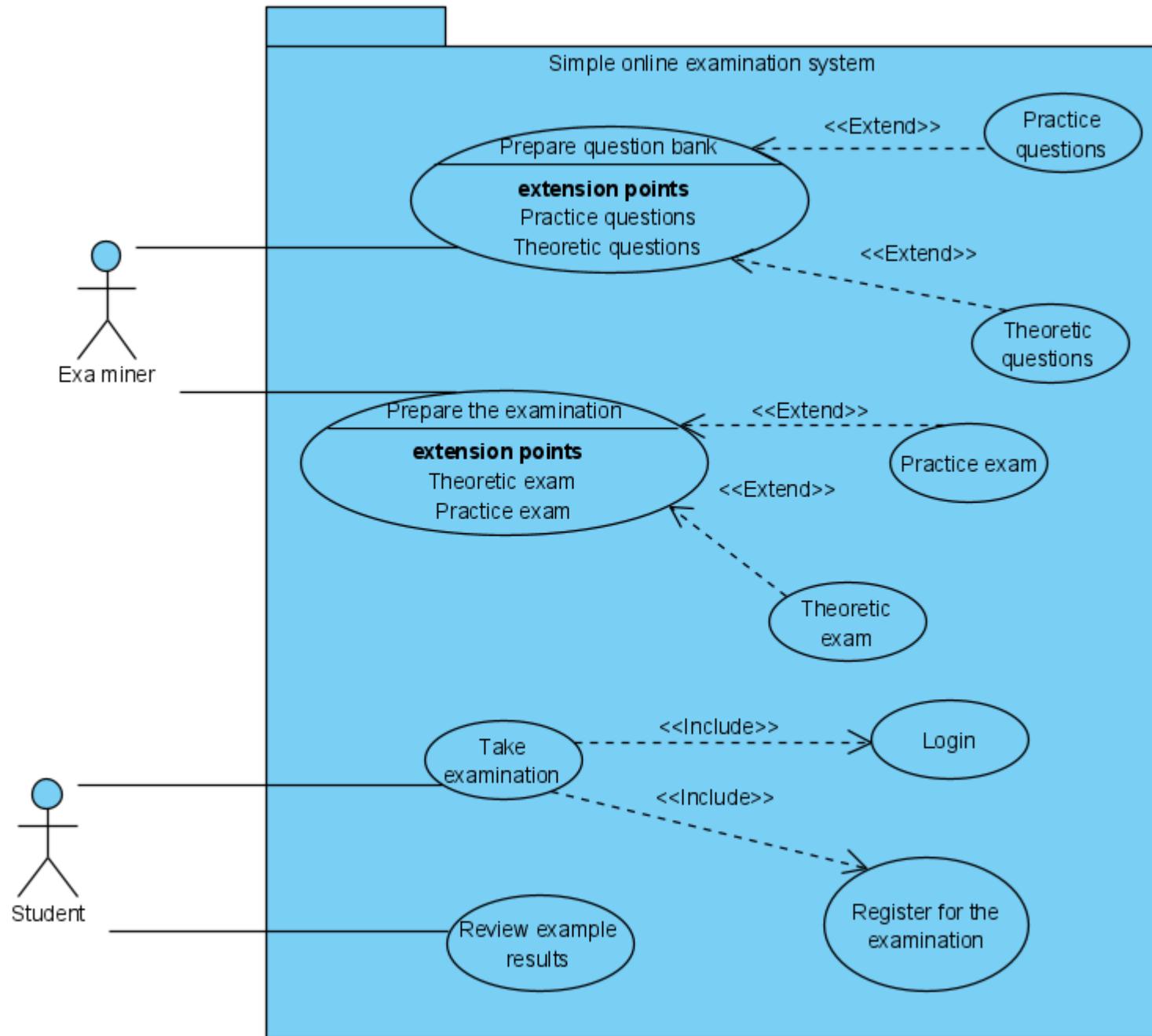
- The student can take an exam and review example results.



Student – use cases (2)

The procedure for taking the exam includes login and registration for the exam.





Example 3

- Task

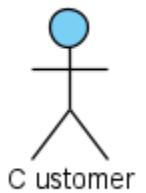
Model the ATM, given by the following description, using the UML use case diagram.

The customer needs money to pay a bill and goes to ATM. Using ATM, the customer can withdraw cash, transfer money, and pay bills. The customer can also check their account balance. All of these functions require login (the correct password). Since the customer can insert invalid password, the ATM login feature has been enhanced to handle invalid passwords.

The customer can withdraw cash with an overdraft protection feature.

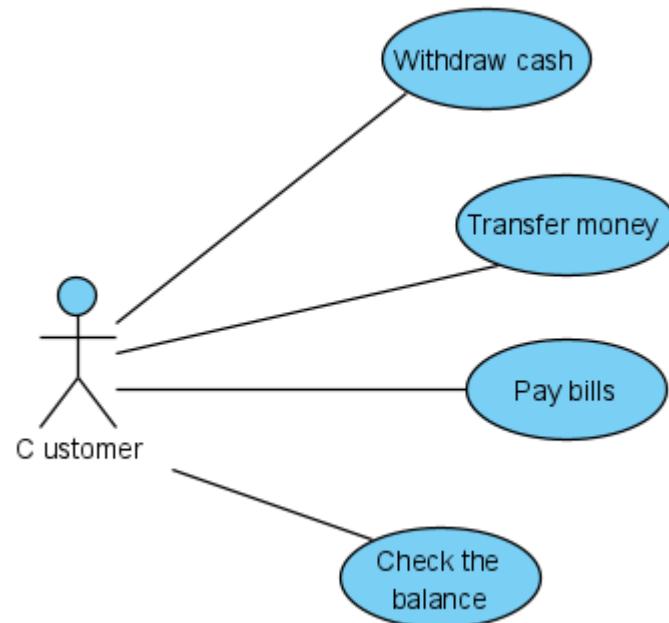
Question

- How many actors we have ?



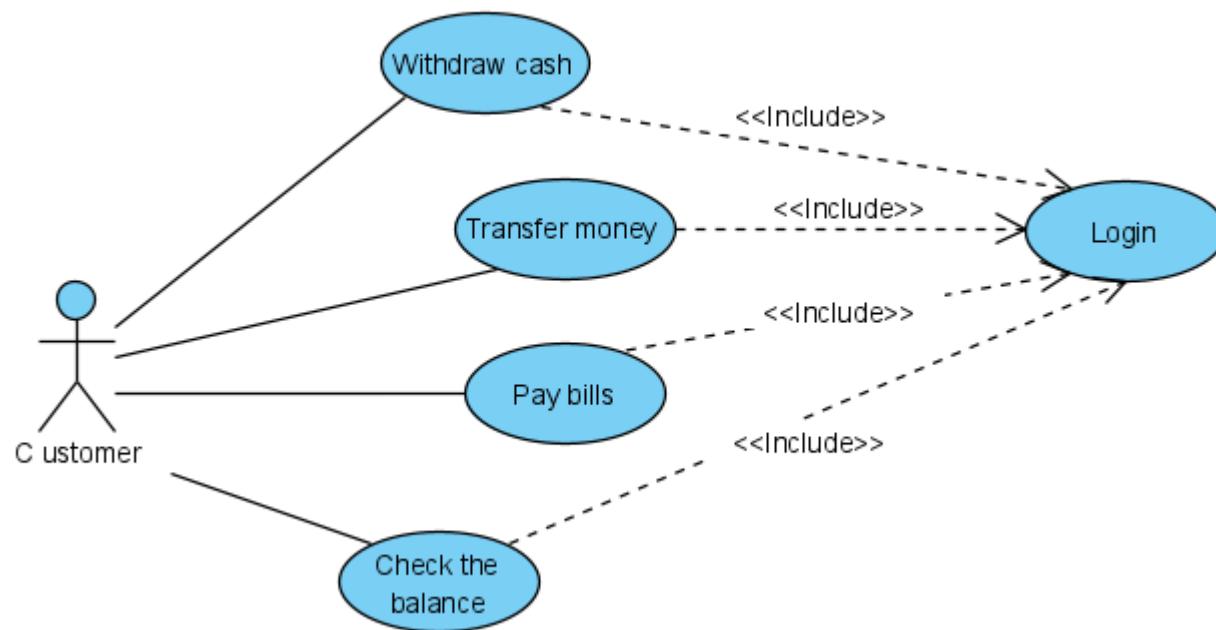
Customer – use cases (1)

- The customer needs money to pay a bill and goes to ATM. Using ATM, the customer can withdraw cash, transfer money, and pay bills. The customer can also check their account balance.



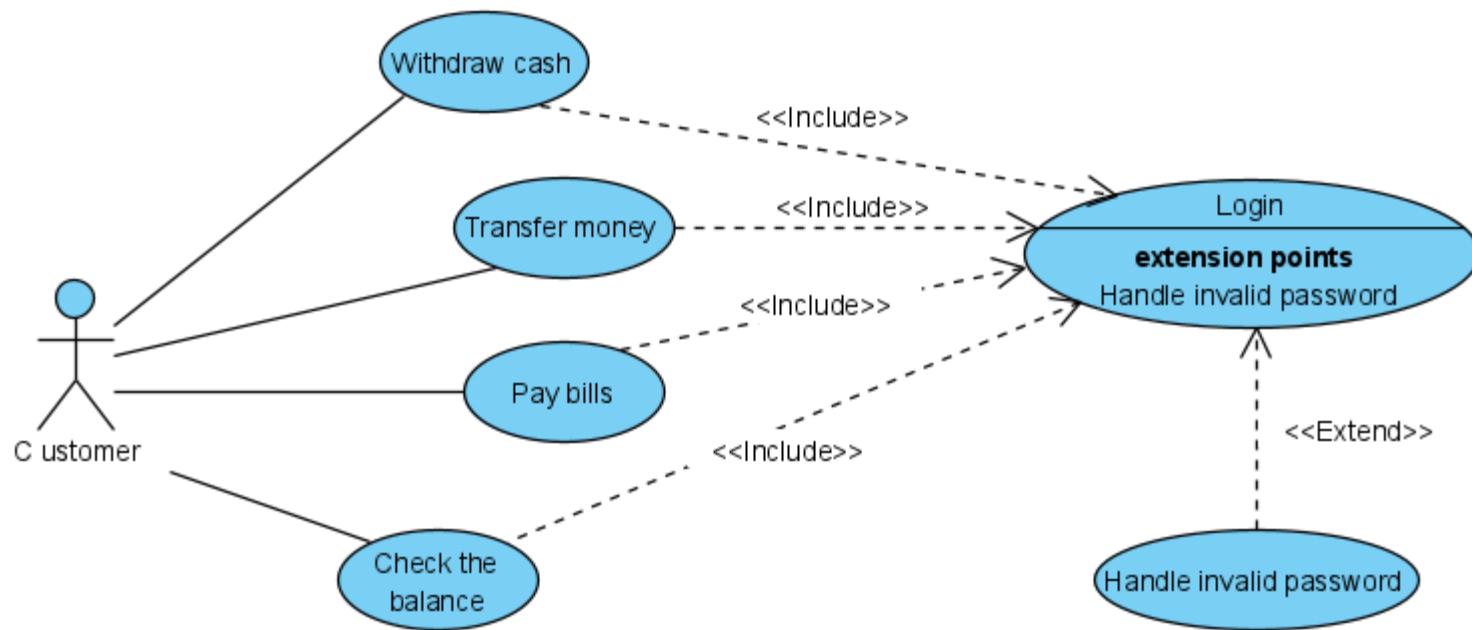
Customer – use cases (2)

- All of these functions require login (the correct password).



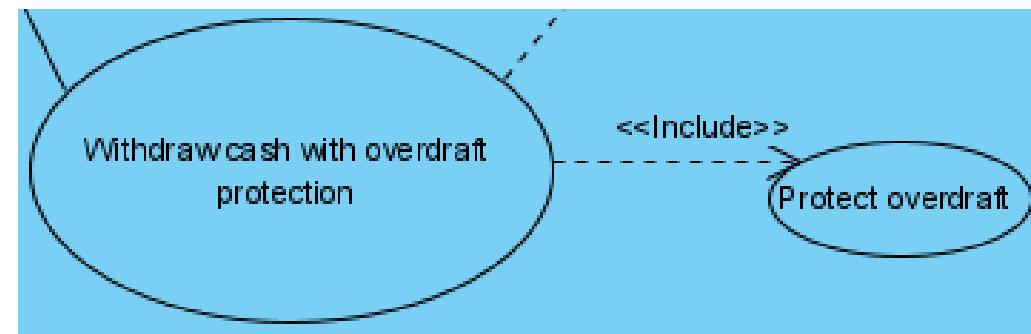
Customer – use cases (3)

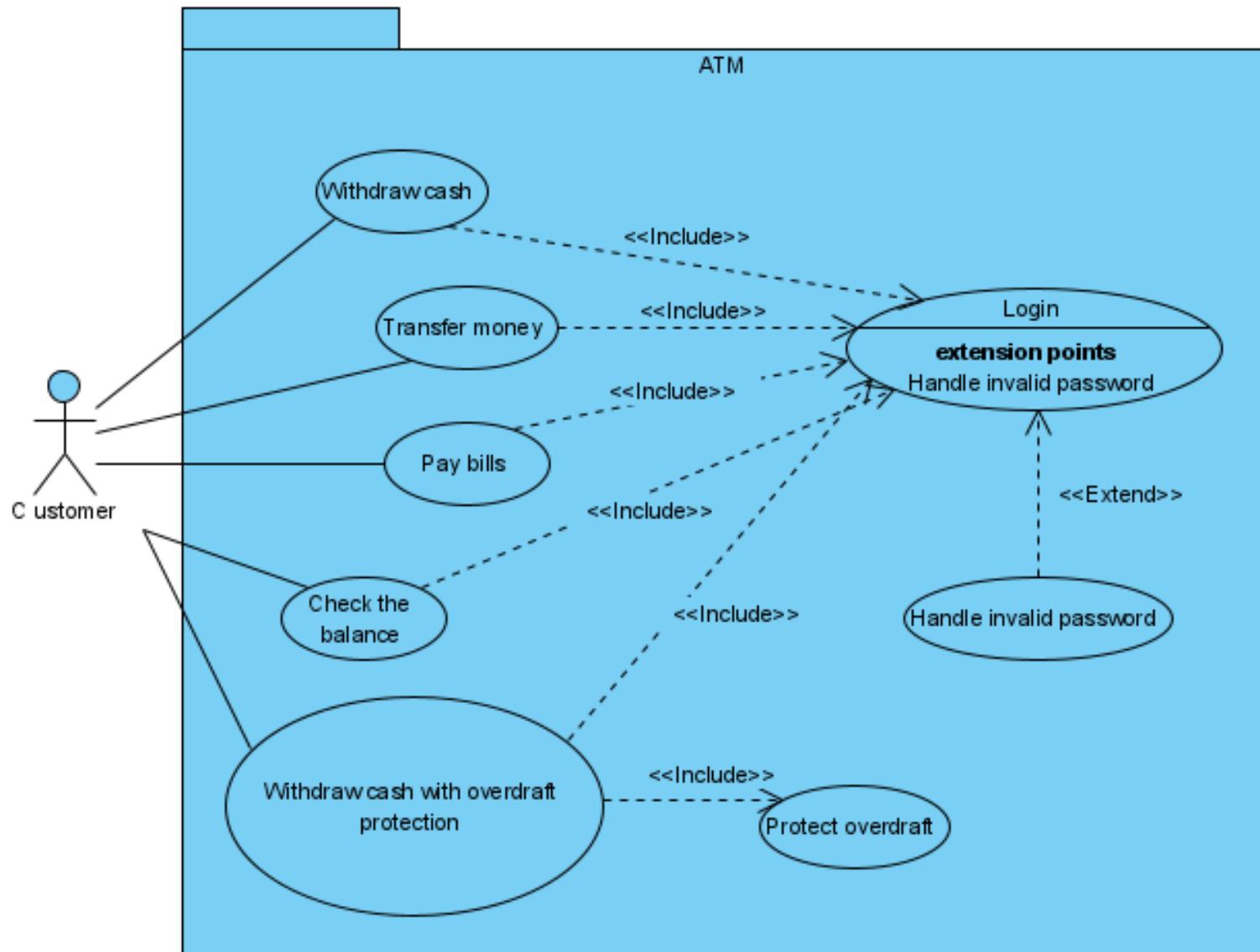
Since the customer can insert invalid password, the ATM login feature has been enhanced to handle invalid passwords.



Customer – use cases (4)

- The customer can withdraw cash with an overdraft protection feature.





Example 4 [2]

- Task

Using UML use case diagrams, model the academic computing cluster.

The users of the cluster are members of the academic community. Users of the cluster can add new jobs for processing, delete existing jobs, and modify the parameters of existing jobs. Changing existing job parameters includes deleting an existing job and creating a new job. Users can also send data to a computer cluster, copy data, and load data from a computer cluster.

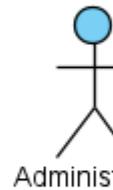
Cluster administrators are responsible for adding new users, changing user data, deleting existing users, and configuring the system. System configuration also includes adjusting Job Scheduler settings and backing up data. Data copy can be performed with data backup or data download. Administrators can also manage jobs and data in a cluster.

Question

- How many actors we have ?



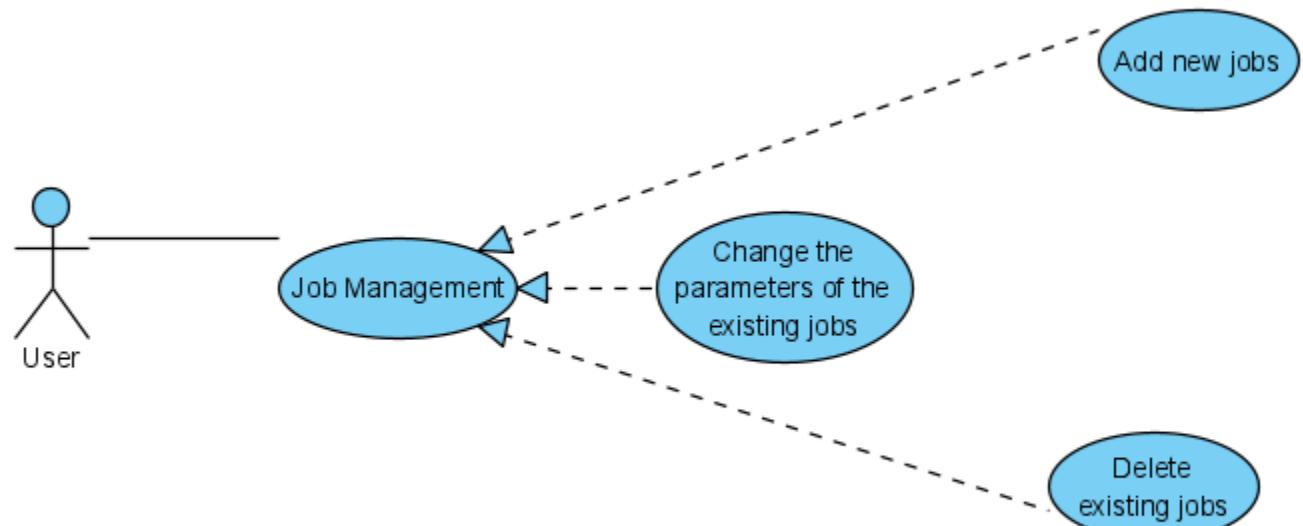
User



Administrator

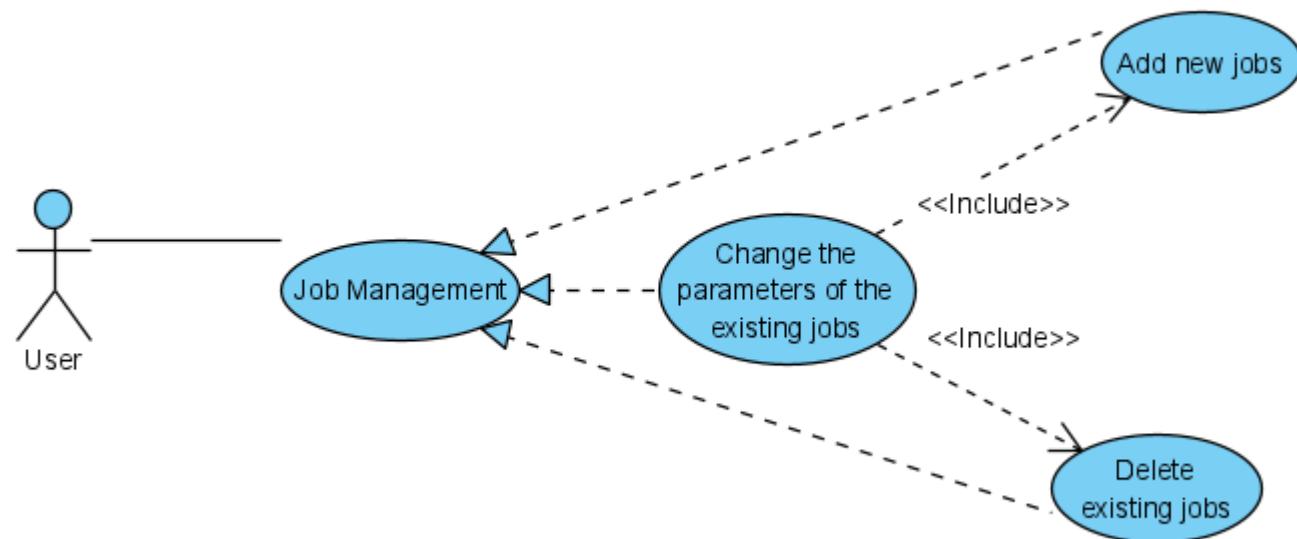
User – use cases (1)

- The users of the cluster are members of the academic community. Users of the cluster can add new jobs for processing, delete existing jobs, and modify the parameters of existing jobs.
- Comment:
 - Although the usage form "Job Management" is not explicitly mentioned in the text, the forms related to the creation, editing and deletion (so-called CRUD operation of the object) are most often displayed as a specialization of the Management form



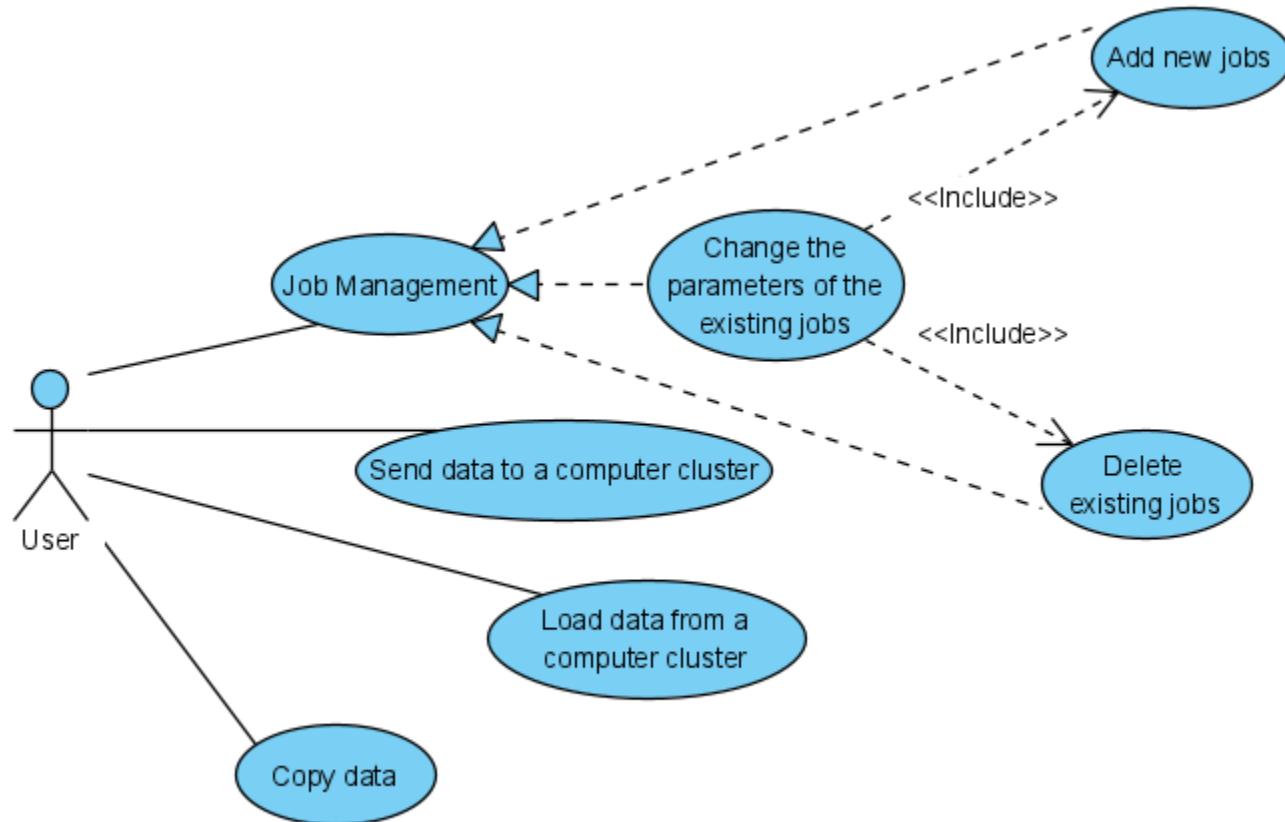
User – use cases (2)

- Changing existing job parameters includes deleting an existing job and creating a new job



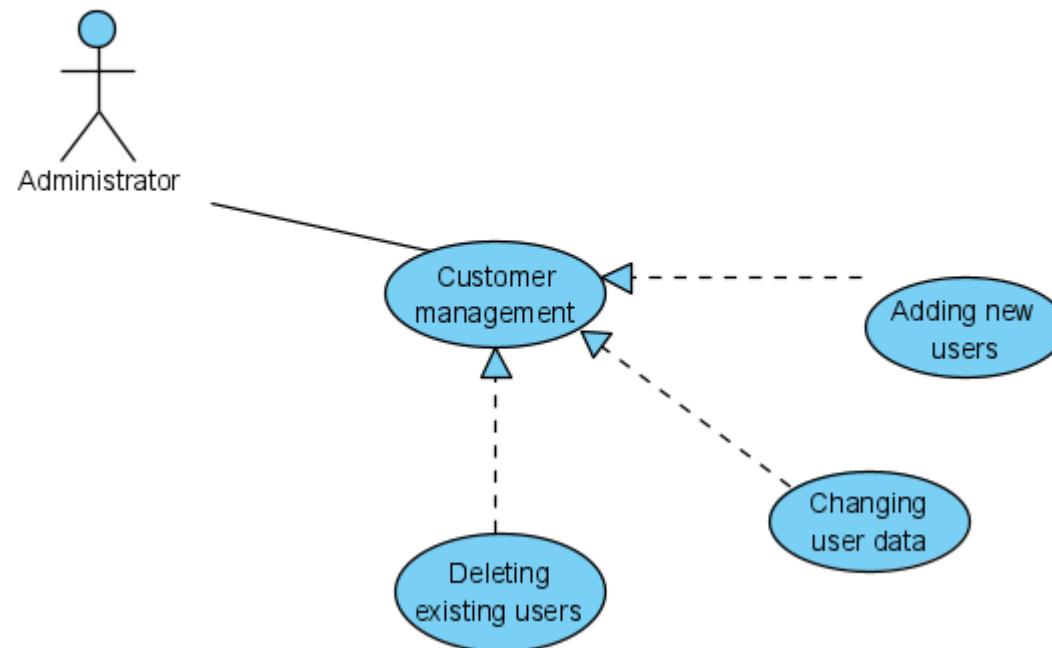
User – use cases (3)

- Users can also send data to a computer cluster, copy data, and load data from a computer cluster.



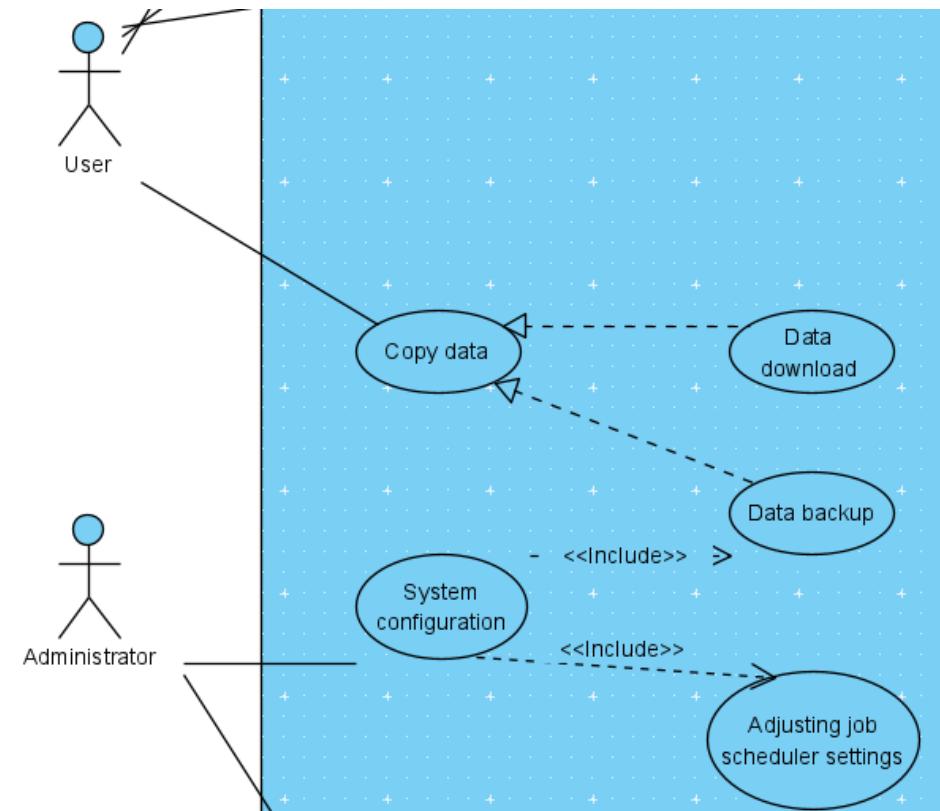
Administrator – use cases (1)

- Cluster administrators are responsible for adding new users, changing user data, deleting existing users, and configuring the system



Administrator – use cases (2)

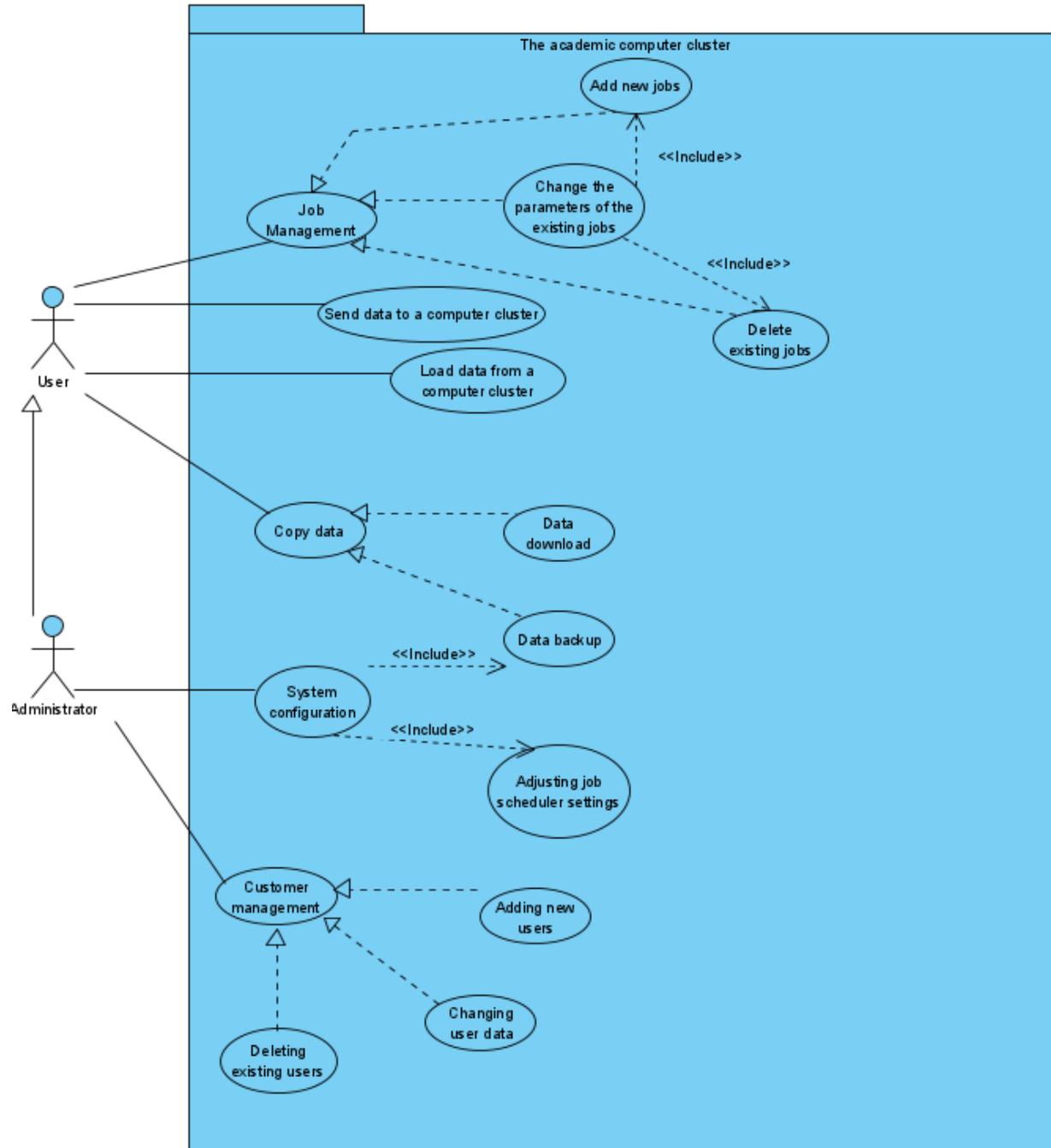
- System configuration also includes adjusting Job Scheduler settings and backing up data. Data copy can be performed with data backup or data download..



Administrator – use cases (3)

- Administrators can also manage jobs and data in a cluster.
- Note: where there is an arrow, the properties of that are inherited





Homework assignment - individually

- User stories + use case diagram (individual).
- Each group member must write a user story and draw a use case diagram for the group's proposed project and upload it to the e-Classroom.
- If you submit the same user stories or the same user diagrams, you will receive 0 points.

UML activity diagram

UML activity diagram – definition

- **Activity diagram** - Depicts the control flow: flow of different activities and actions.
- Activity diagram is an UML variant of flowchart.
- Models a control flow at SW execution.

Activity Diagram Notation

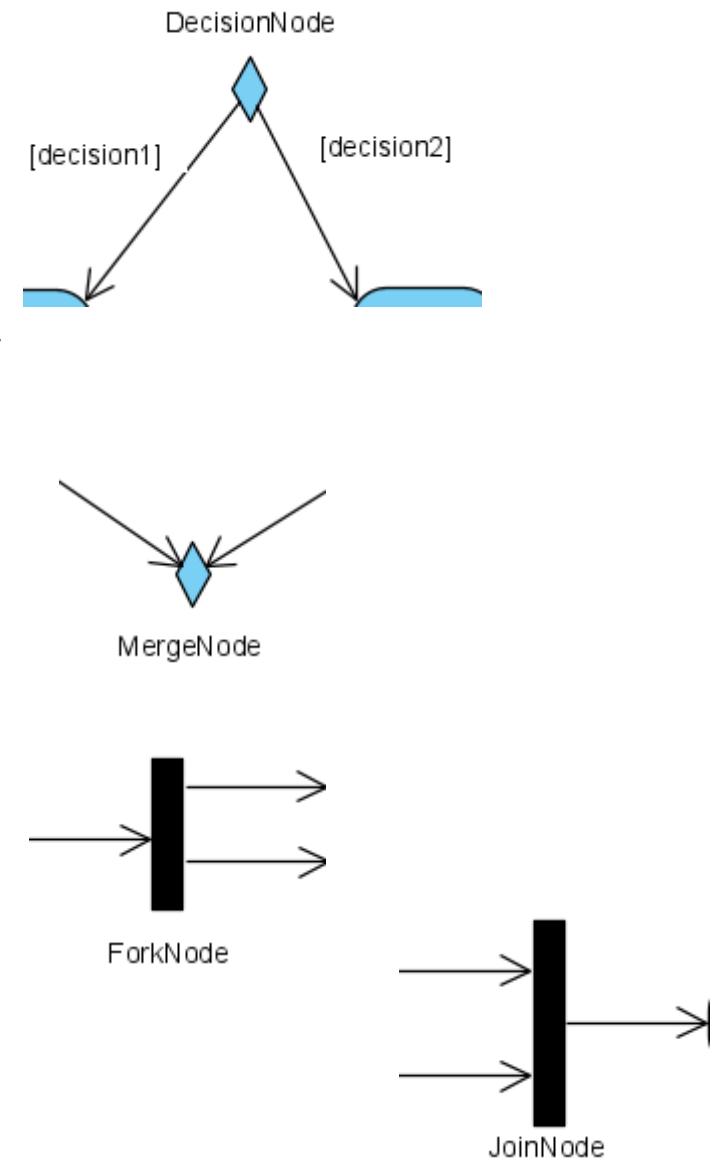
Notation [8]

- Activity - Is used to represent a set of actions
- Action - A task to be performed
- Control Flow - Shows the sequence of execution
- Object Flow - Show the flow of an object from one activity (or action) to another activity (or action).
- Initial Node - Portrays the beginning of a set of actions or activities
- Activity Final Node - Stop all control flows and object flows in an activity (or action)
- Object Node - Represent an object that is connected to a set of Object Flows



Notation [8]

- Decision Node - Represent a test condition to ensure that the control flow or object flow only goes down one path
- Merge Node - Bring back together different decision paths that were created using a decision-node.
- Fork Node - Split behavior into a set of parallel or concurrent flows of activities (or actions)
- Join Node - Bring back together a set of parallel or concurrent flows of activities (or actions).



Notation [8]

- **Swimlane and Partition** - A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread

Partition	Partition2

Example 1. [9]

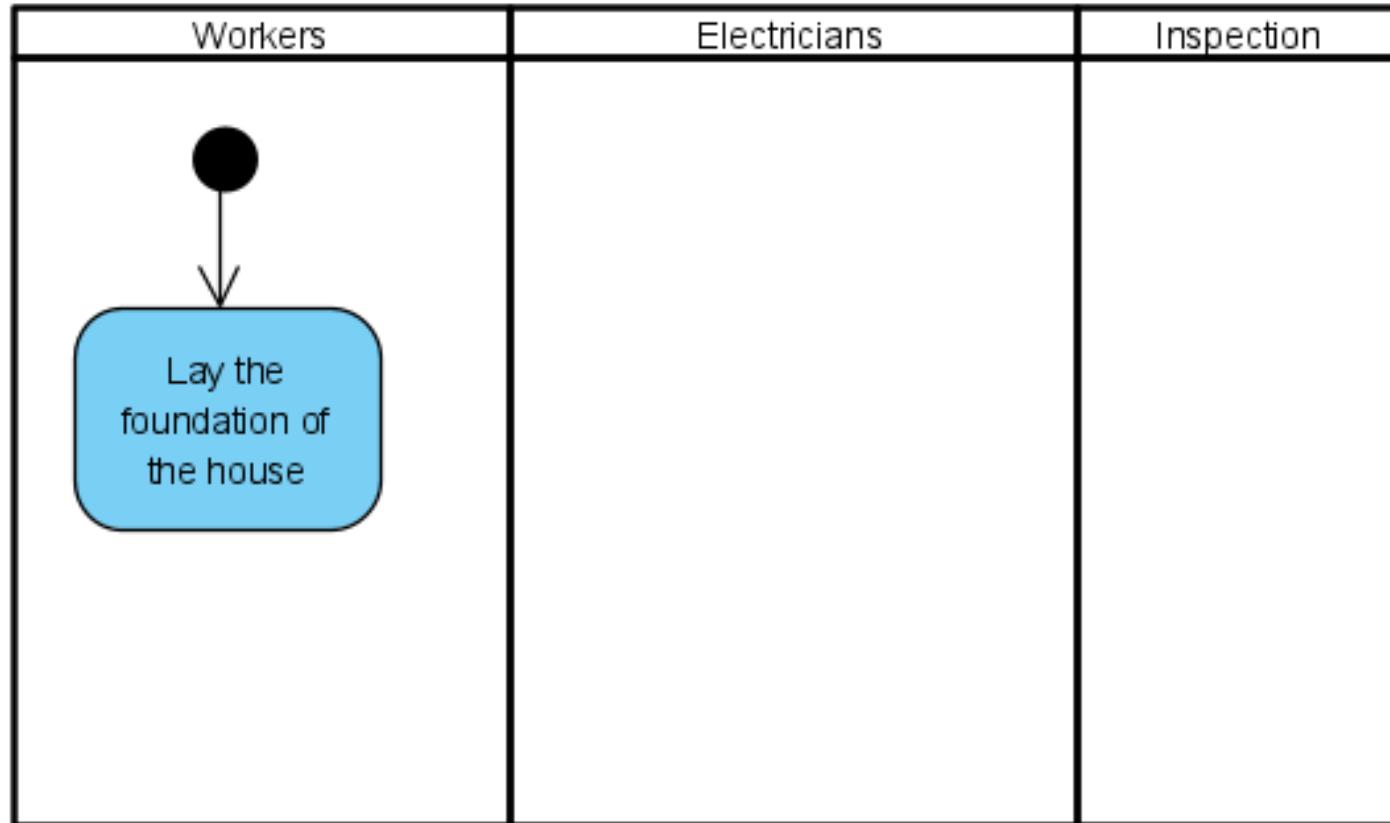
- Model an activity diagram for building a house. First, workers lay the foundation of the house. Then, in parallel, the workers build the frame (floors and load-bearing walls) while the electrician installs the electrical installations in the foundations. Then the electrician installs the electrical installations in the load-bearing walls. While some workers install the roof, others install the partition walls. After that, the electrician installs other electrical installations throughout the house. After that, workers perform exterior and interior plastering. Finally, workers install doors and windows. During the inspection, the condition of the house is assessed.

Swimlanes

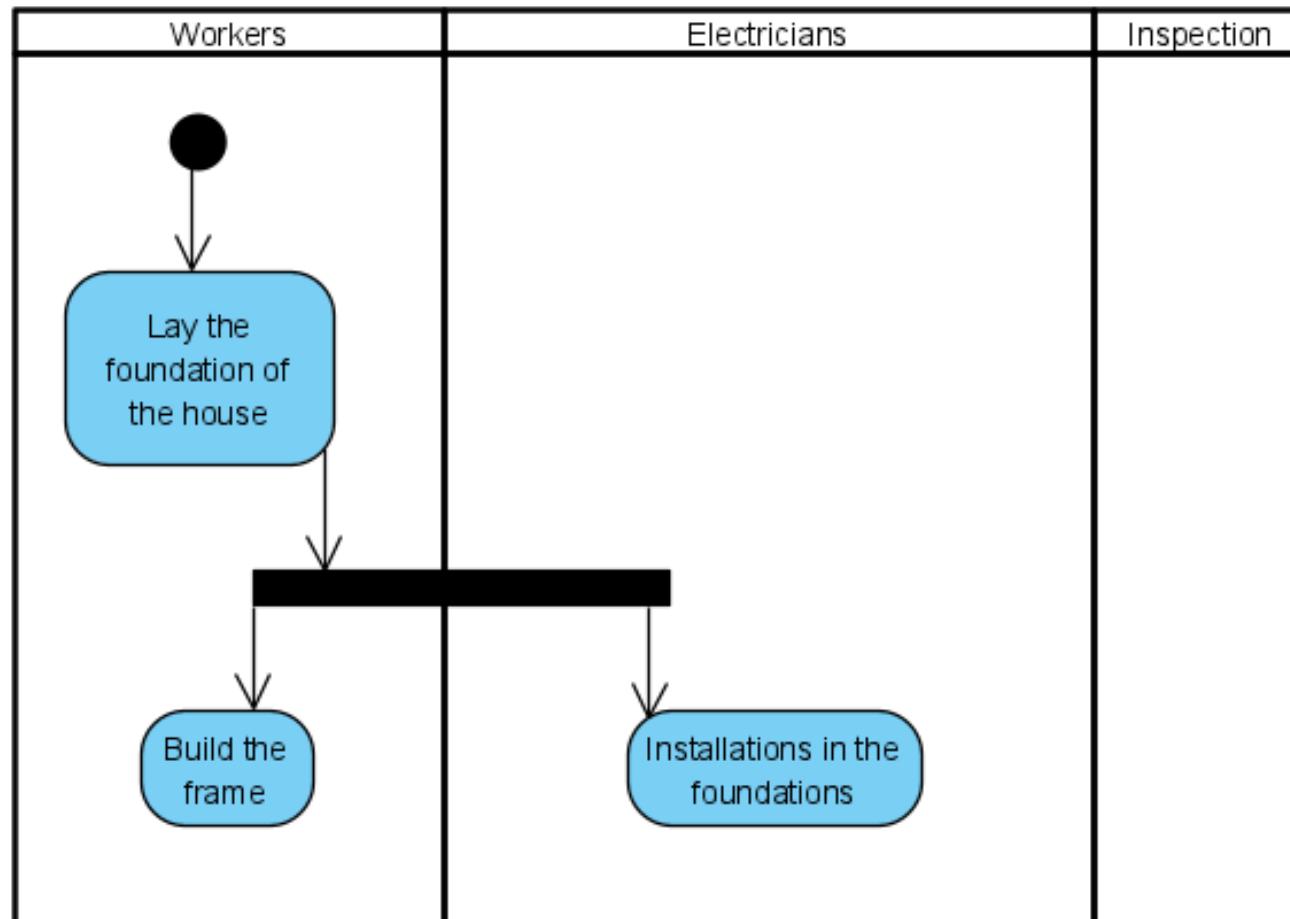
- The activity is divided into three swimlanes. The first is for the workers, the second for the electricians, and the third for inspection.

Workers	Electricians	Inspection

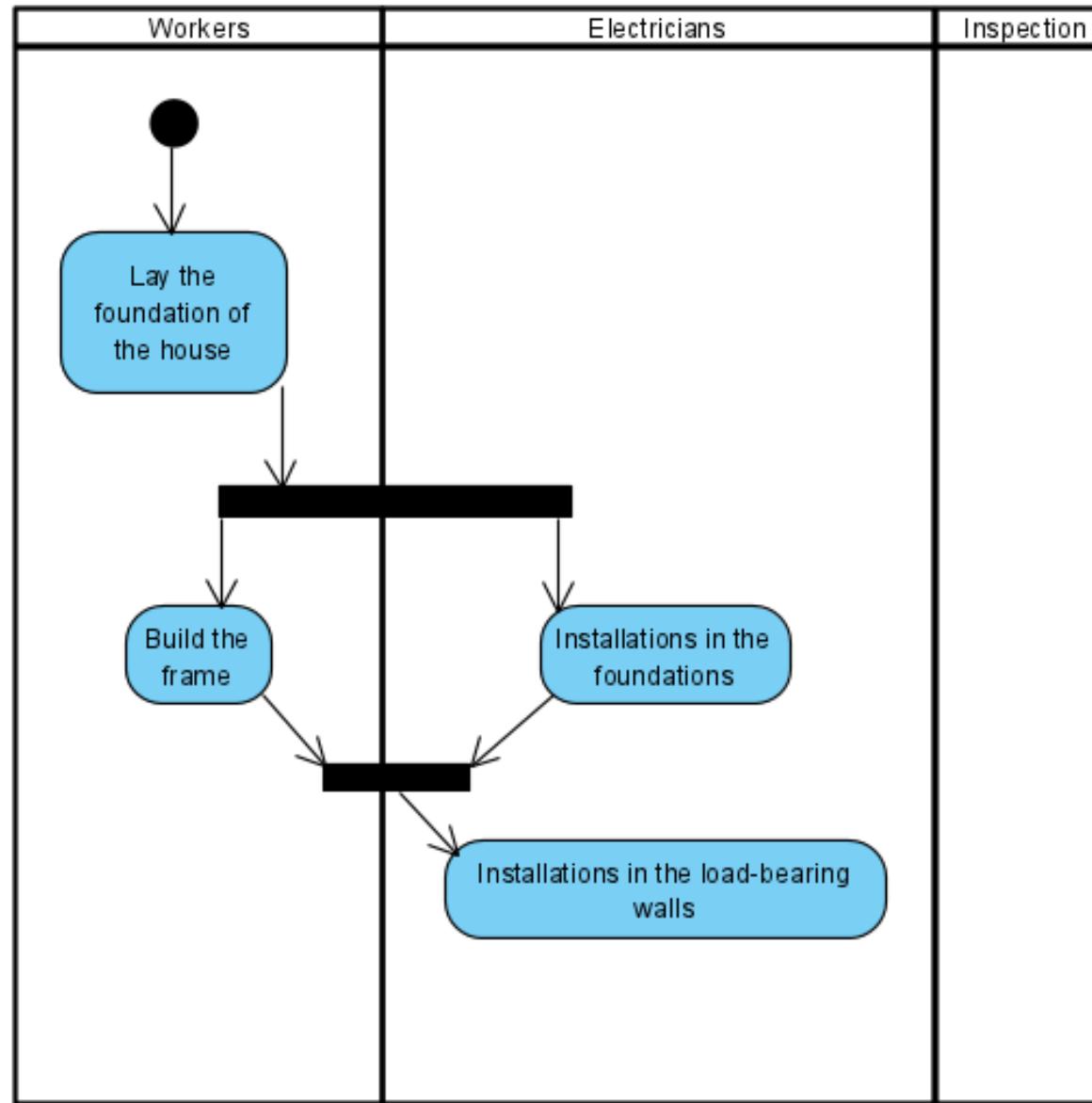
- First, workers lay the foundation of the house.



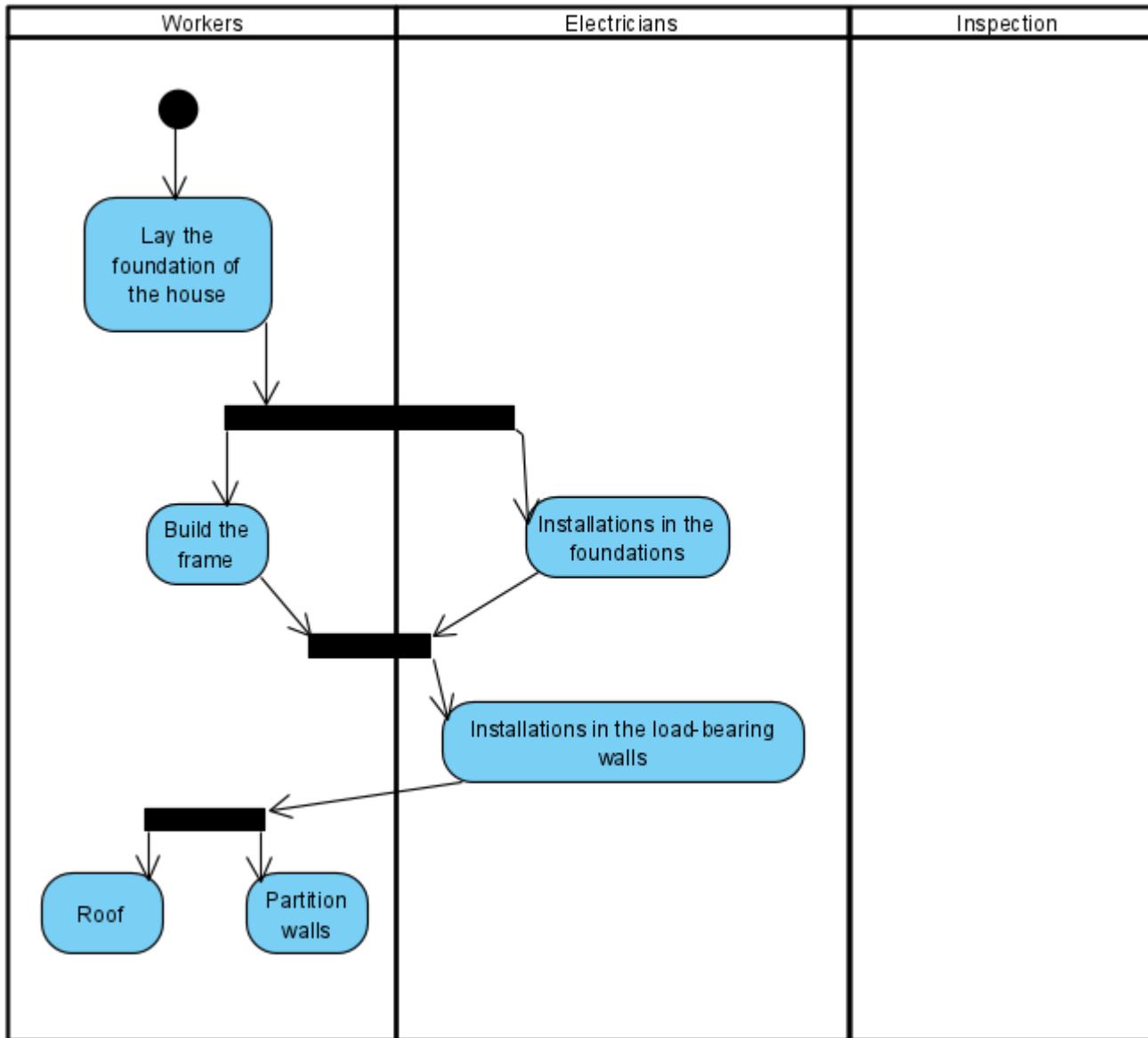
- Then, in parallel, the workers build the frame (floors and load-bearing walls) while the electrician installs the electrical installations in the foundations.



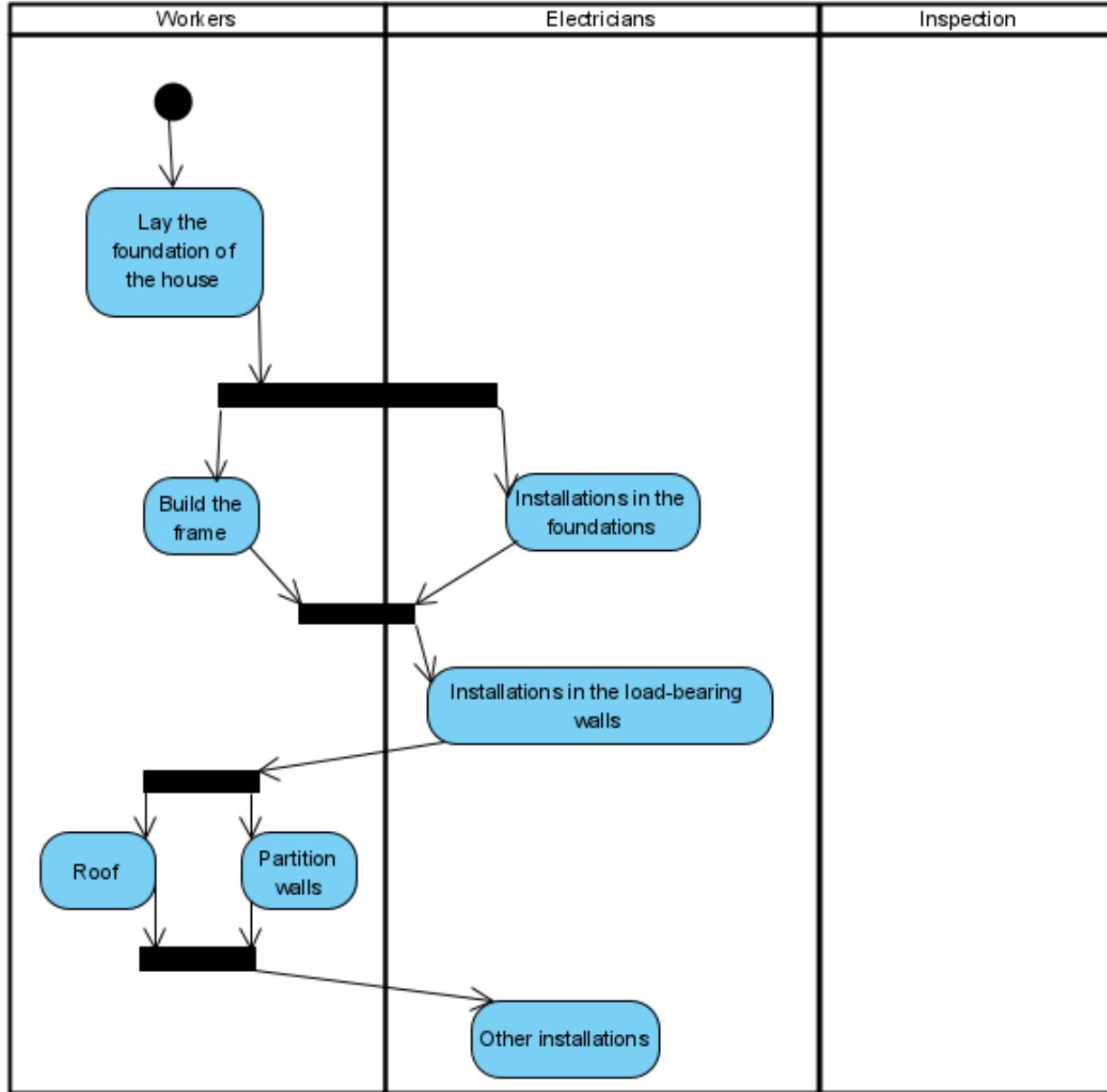
- Then the electrician installs the electrical installations in the load-bearing walls.



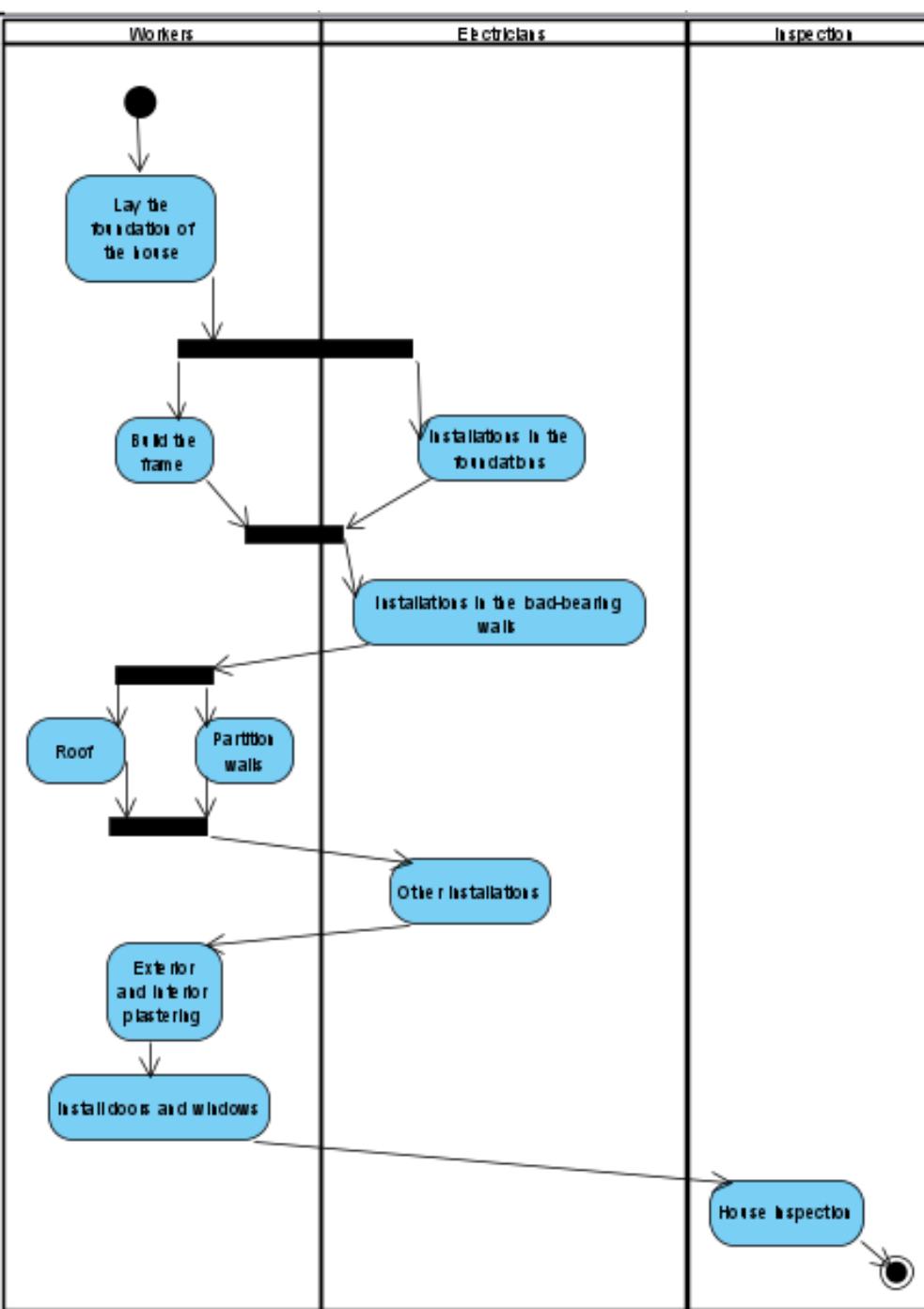
- While some workers install the roof, others install the partition walls.



- After that, the electrician installs other electrical installations throughout the house.



- After that, workers perform exterior and interior plastering. Finally, workers install doors and windows. During the inspection, the condition of the house is assessed.

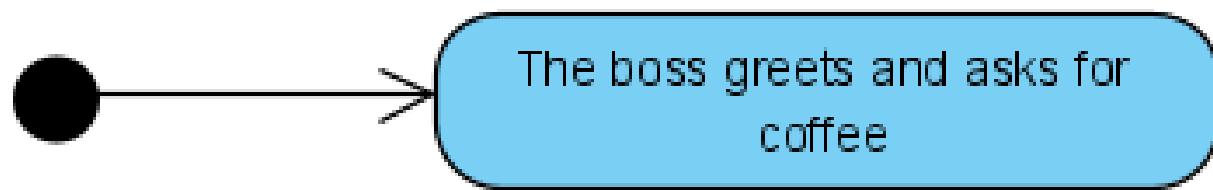


Example 2. [9]

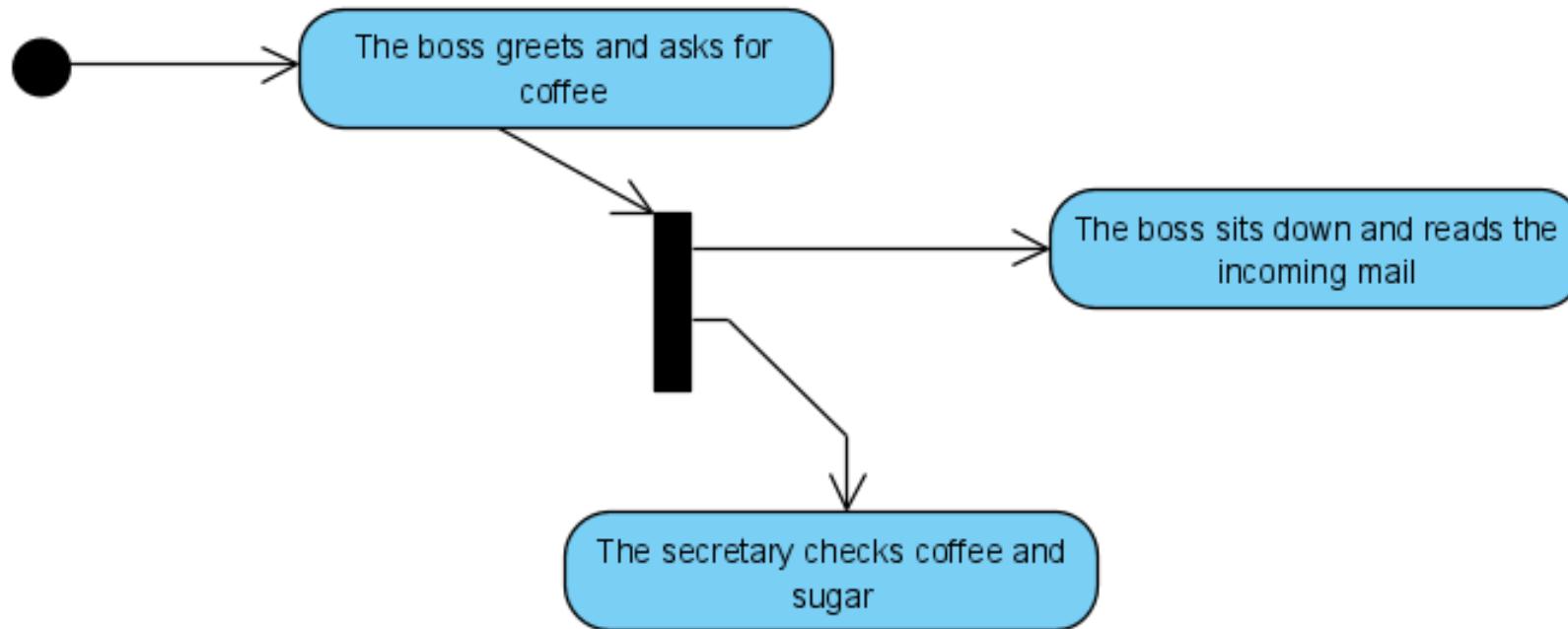
Model the process of making coffee in a company with an activity diagram. Suppose that in the company in question, the secretary makes coffee for the boss every morning. The sequence of activities is as follows. Upon arrival, the boss greets the secretary and asks her for coffee. In addition, the boss sits down and reads the incoming mail. During this time, the secretary first checks if there is enough coffee and sugar in the office. If there is enough coffee and sugar, then put water on to boil. If there is not enough coffee or sugar, the secretary goes to the company's procurement department. The secretary asks for coffee and sugar. If the procurement department has coffee or sugar, then they give it to her and she goes back to the office and boil the coffee water. If the procurement department does not have coffee or sugar, she orders it to be replenished as soon as possible. Since the secretary knows that the boss can not do without her morning coffee, she stays with the procurement department until she gets her coffee. As soon as he gets his coffee, the secretary goes back to the office and makes coffee. In the meantime, if more than 45 minutes have passed since the boss ordered the coffee, the boss stops reading the mail and leaves the office in a huff. In any case, the secretary prepares the coffee as soon as she has the coffee and sugar and the water boiling, even if the boss has already left. If the boss is still there, the boss drinks coffee.

Remark: create this diagram without swimlanes

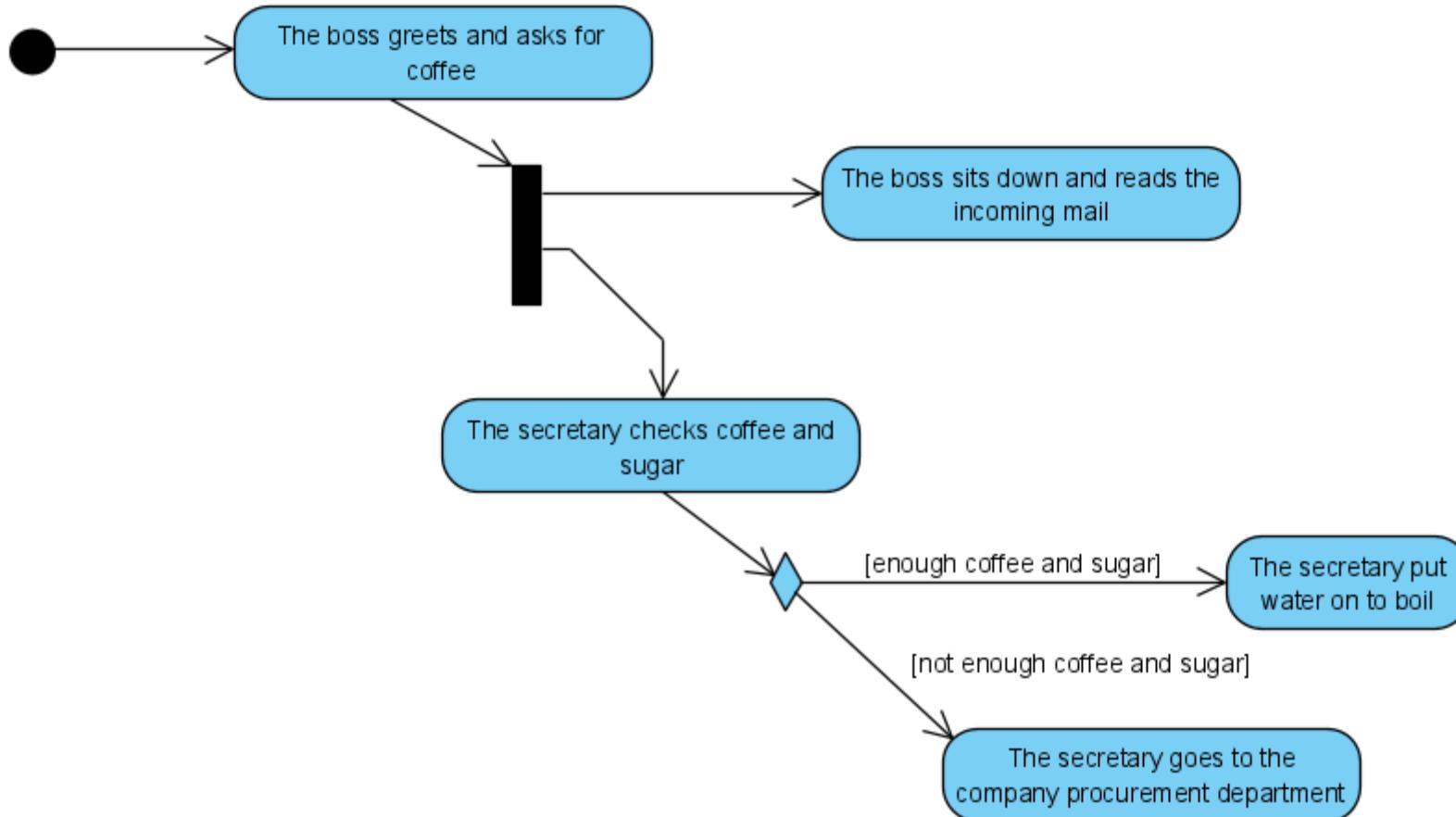
- Model the process of making coffee in a company with an activity diagram. Suppose that in the company in question, the secretary makes coffee for the boss every morning. The sequence of activities is as follows. Upon arrival, the boss greets the secretary and asks her for coffee.



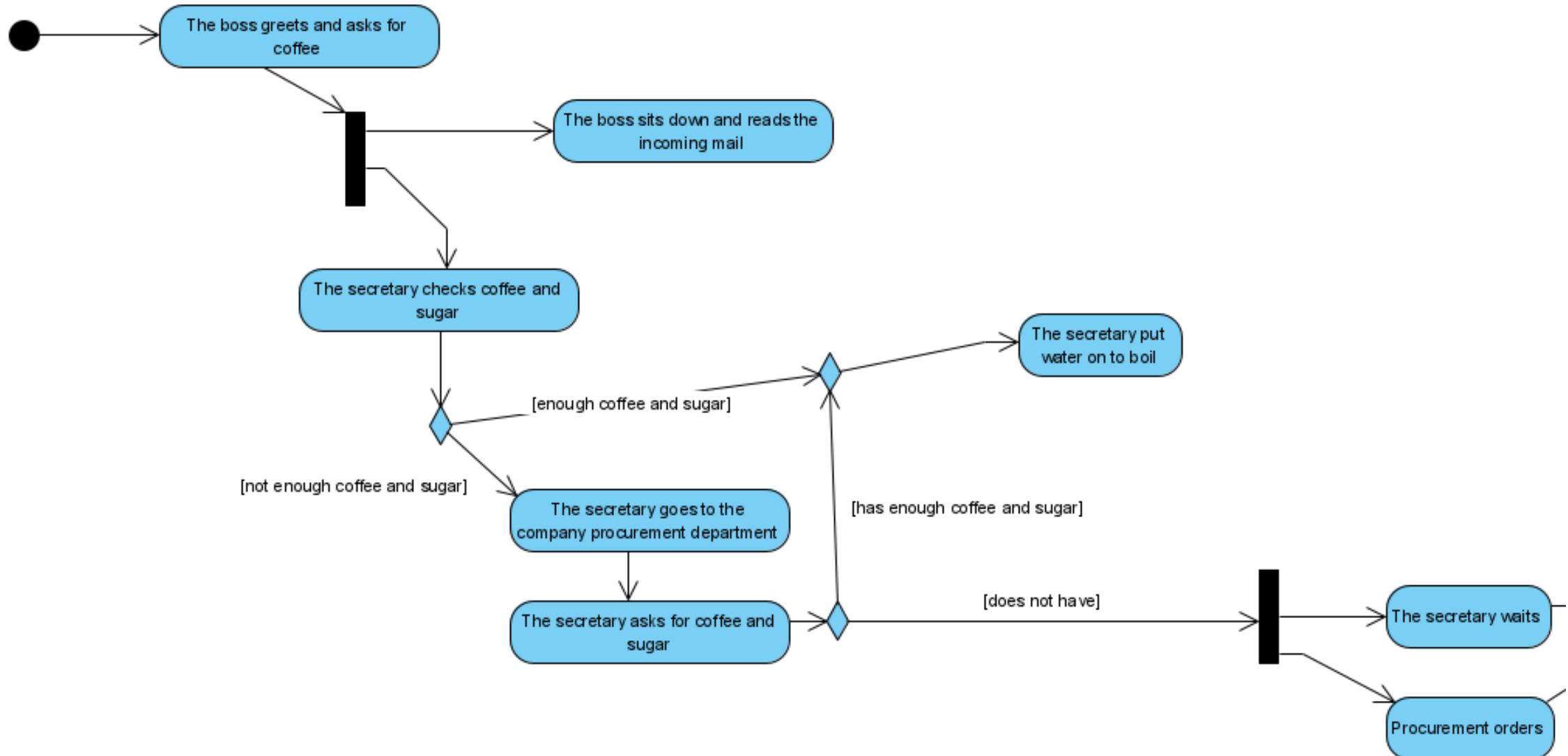
- In addition, the boss sits down and reads the incoming mail. During this time, the secretary first checks if there is enough coffee and sugar in the office.



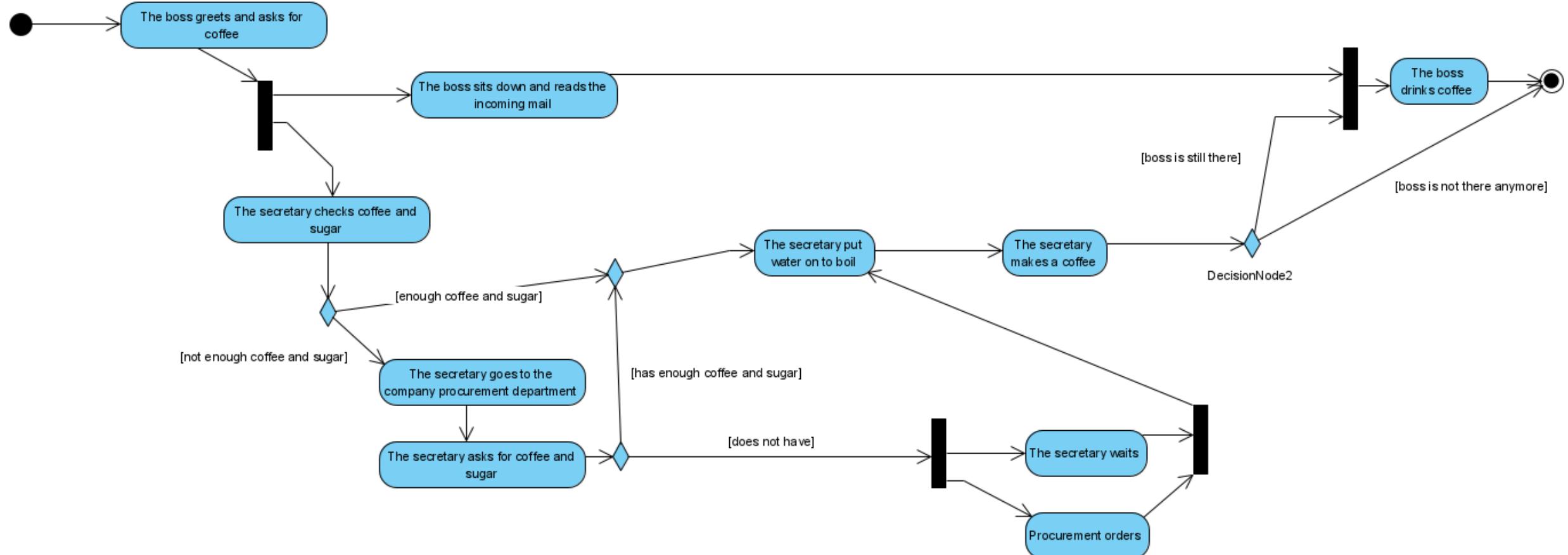
- **If there is** enough coffee and sugar, then put water on to boil. **If there is not** enough coffee or sugar, the secretary goes to the company's procurement department.



- The secretary asks for coffee and sugar. If the procurement department has coffee or sugar, then they give it to her and she goes back to the office and boil the coffee water. If the procurement department does not have coffee or sugar, she orders it to be replenished as soon as possible. Since the secretary knows that the boss can not do without her morning coffee, she stays with the procurement department until she gets her coffee.



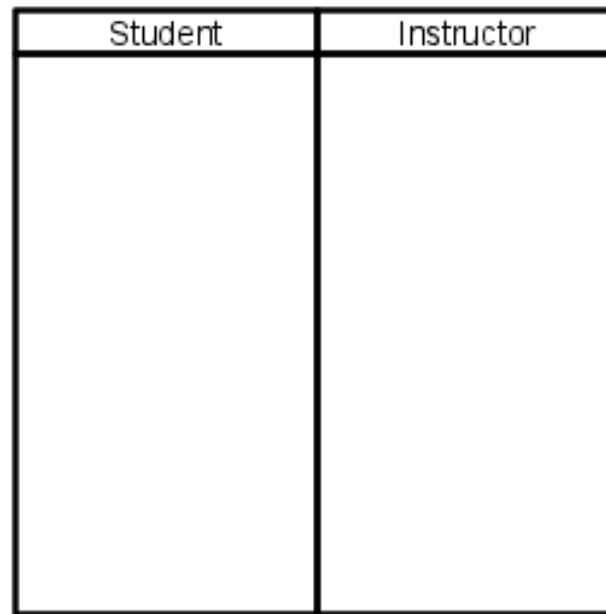
- As soon as he gets his coffee, the secretary goes back to the office and **makes coffee**. **In the meantime, if more than 45 minutes** have passed since the boss ordered the coffee, the boss stops reading the mail and **leaves the office in a huff**. **In any case**, the secretary prepares the coffee as soon as she has the coffee and sugar and the water boiling, **even if the boss has already left**. If the boss is still there, **the boss drinks coffee**.



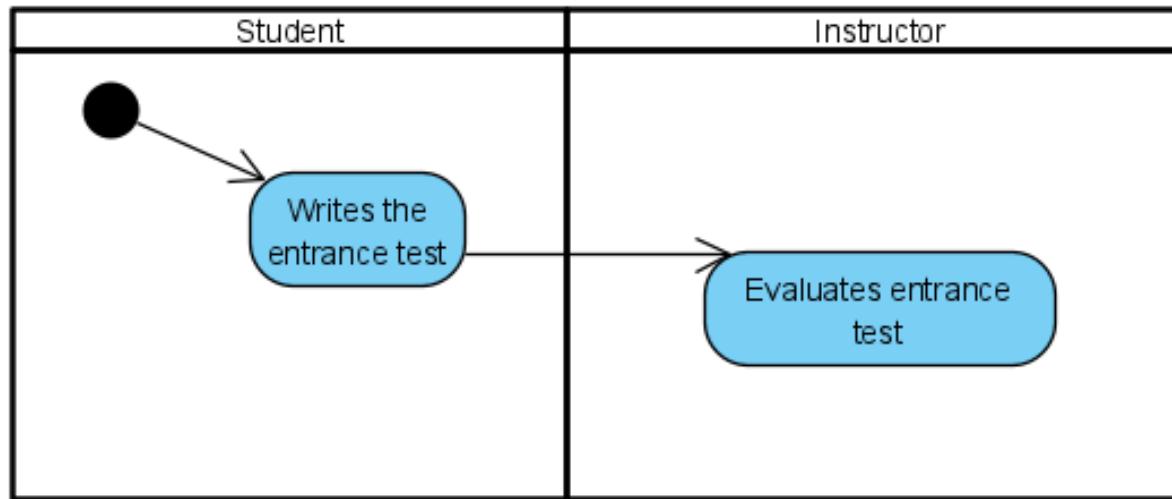
Example 3. [9]

- Illustrate your participation in the Java Programming Basics skill with an activity diagram. The student wants to enroll and take a Java course. The first step is to pass the entrance test. The entrance test is graded by the instructor. If the student passes the entrance test, he/she will be enrolled in the Java course. Java courses are structured in such a way that the instructor speaks first and the students listen to him. At the end of the course, the lecturer gives homework to the students. The student writes the homework for the next week and submits it to the lecturer. At the same time, the lecturer assesses the homework and the student considers whether he has written the homework well enough (he bites his nails).
- The lecturer makes a decision on whether the homework was good enough. If he is satisfied with the homework, the procedure is repeated and the student comes to the Java class again. So if a student has managed to complete 10 homework assignments, he has passed the skill. In all other cases, the student is considered to have failed.
- Remark: create this diagram using swimlanes

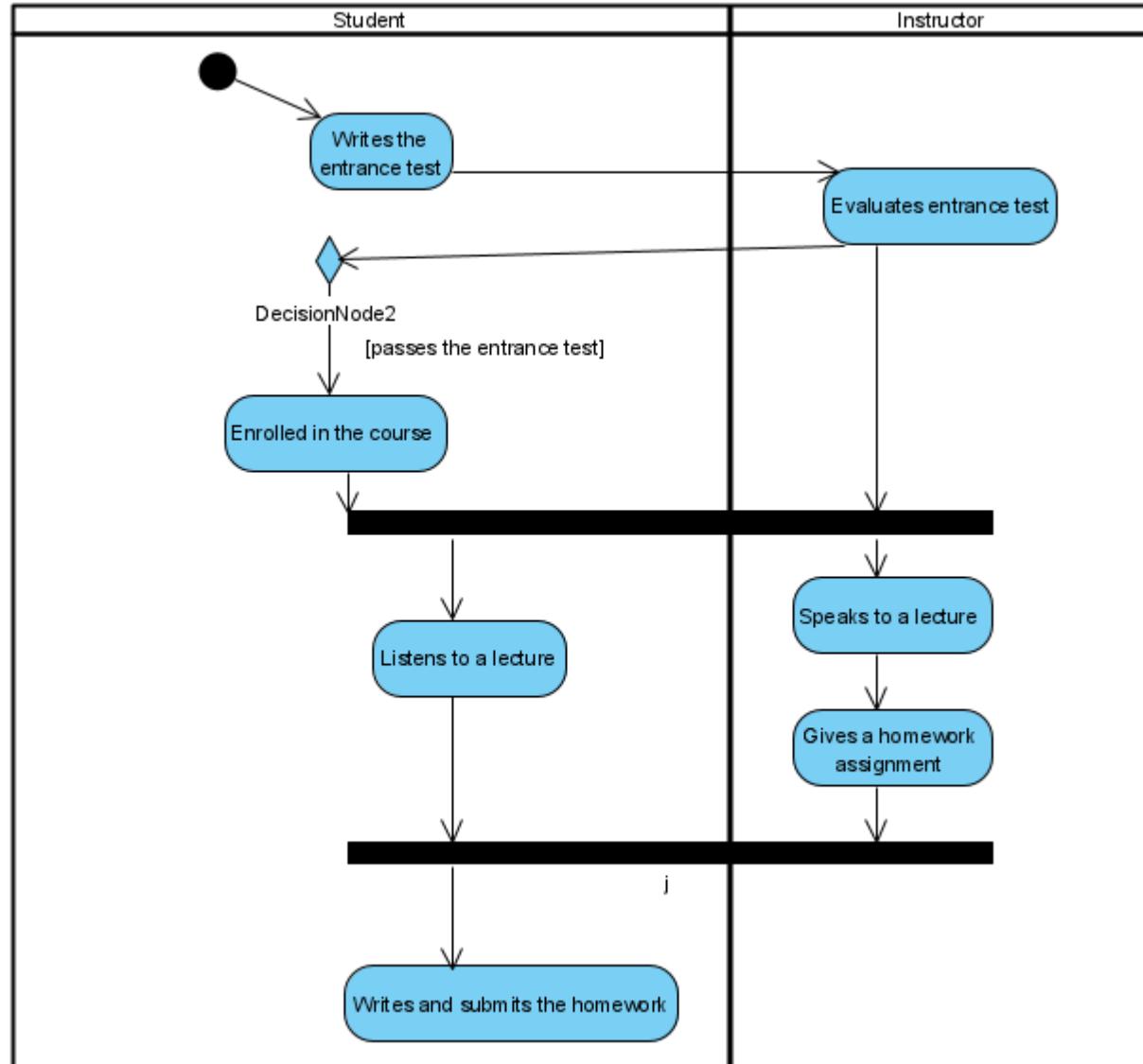
Swimlanes



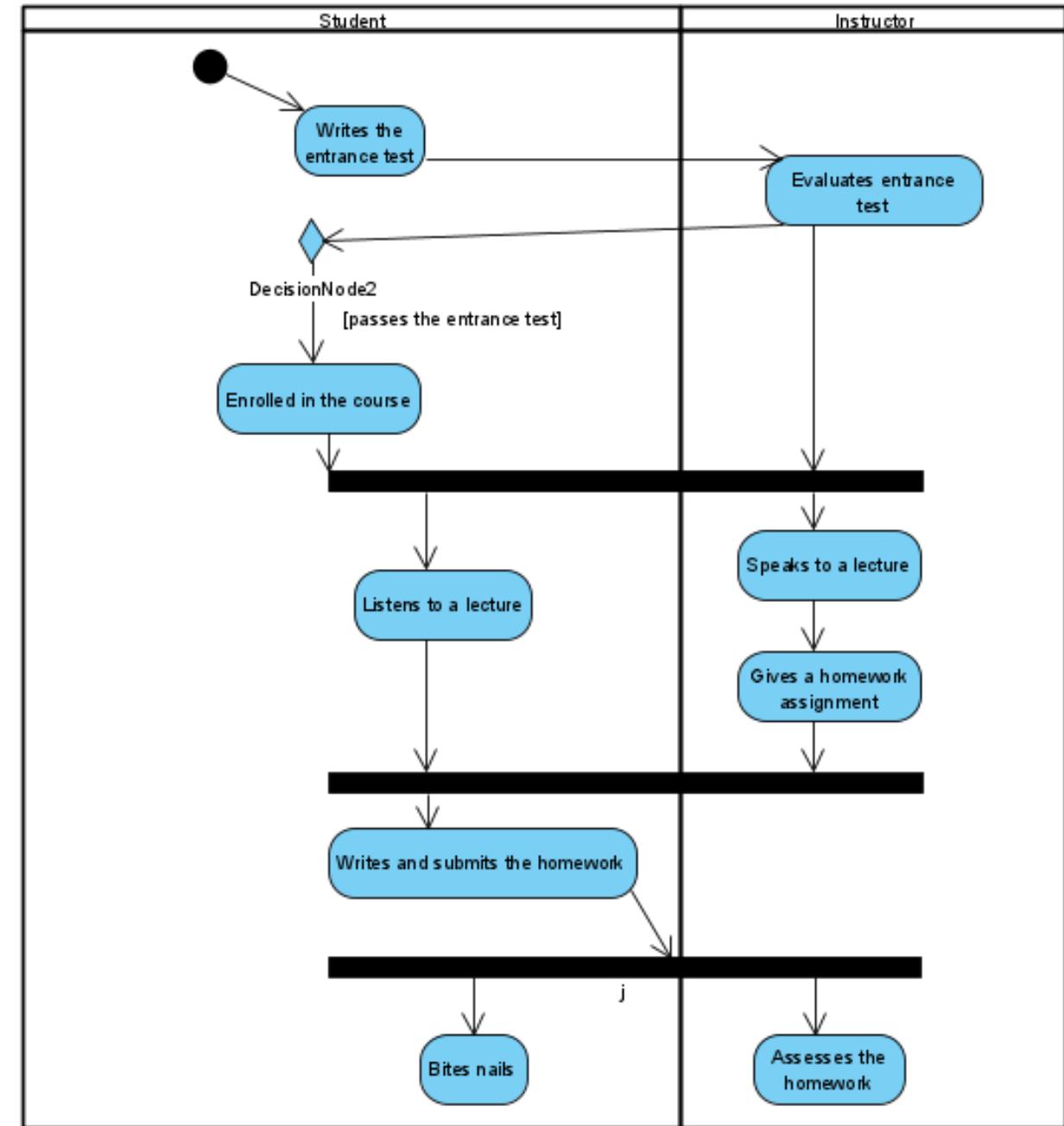
- The student wants to enroll and take a Java course. The first step is to pass the entrance test. The entrance test is graded by the instructor.



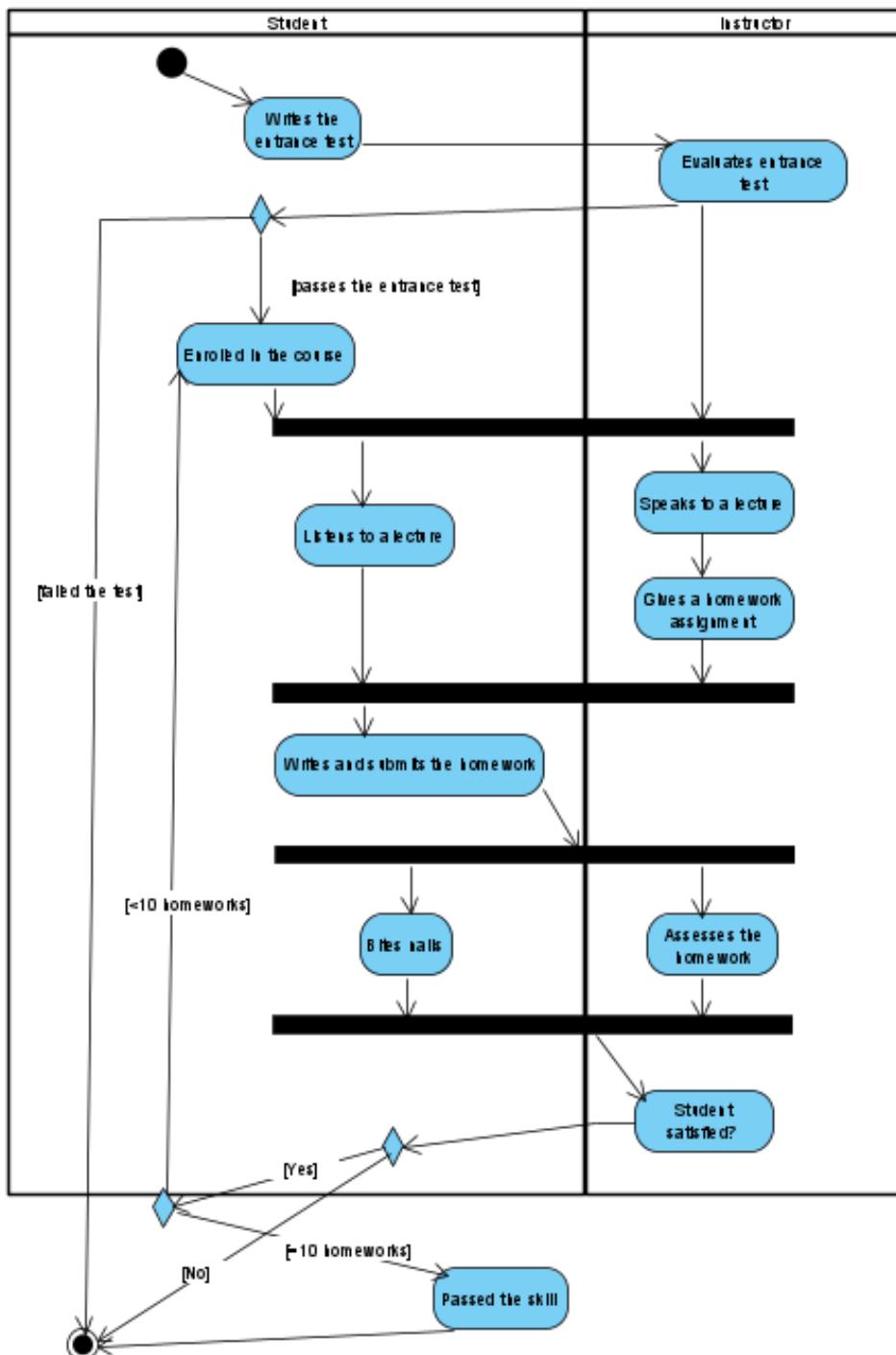
- **If the student passes the entrance test, he/she will be enrolled in the Java course.**
 Java courses are structured in such a way that the **instructor speaks first and the students listen to him**. At the end of the course, **the lecturer gives homework to the students**. The student writes the homework for the next week and submits it to the lecturer.



- At the same time, the lecturer assesses the homework and the student considers whether he has written the homework well enough (he bites his nails).



- The lecturer makes a decision on whether the homework was good enough. If he is satisfied with the homework, the procedure is repeated and the student comes to the Java class again. So if a student has managed to complete 10 homework assignments, he has passed the skill. In all other cases, the student is considered to have failed.



UML sequence diagram

UML sequence diagram - definition

- **Sequence diagram** - describes the sequence of messages and interactions that happen between actors and objects.

Dimensions in sequence diagram [5]

- Object dimension
 - The horizontal axis shows the elements that are involved in the interaction
 - The objects involved in the operation are listed from left to right according to when they take part in the message sequence
- Time dimension
 - The vertical axis represents time proceedings (or progressing) down the page
- Note:
 - Time in a sequence diagram is all about ordering, not duration.

Sequence Diagram Notation

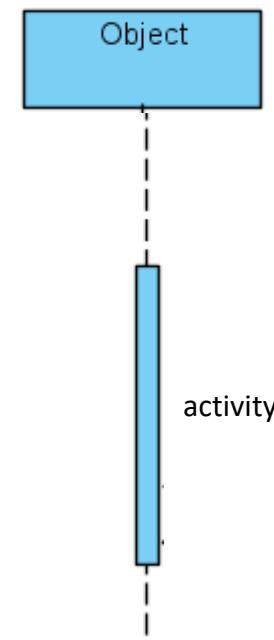
Actor [5]

- a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data)
- represent roles played by human users, external hardware, or other subjects.



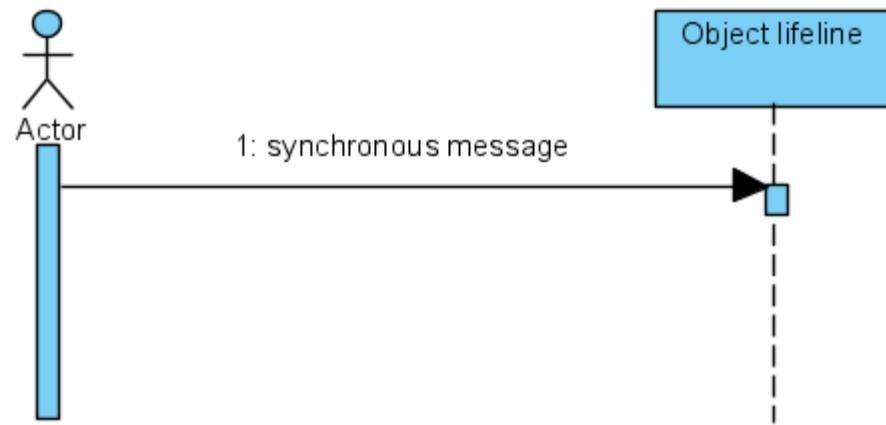
Lifeline Notation [5,6]

- Lifeline - represent the different objects or parts that interact with each other in the system during the sequence, or in other words represents an individual participant in the Interaction.



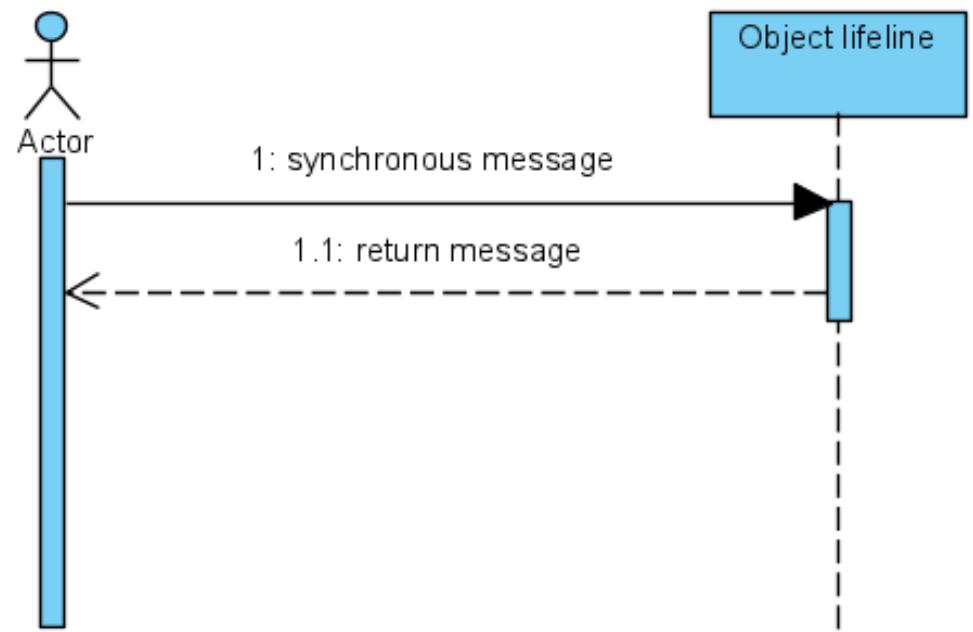
Synchronous message [6]

- Synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one.



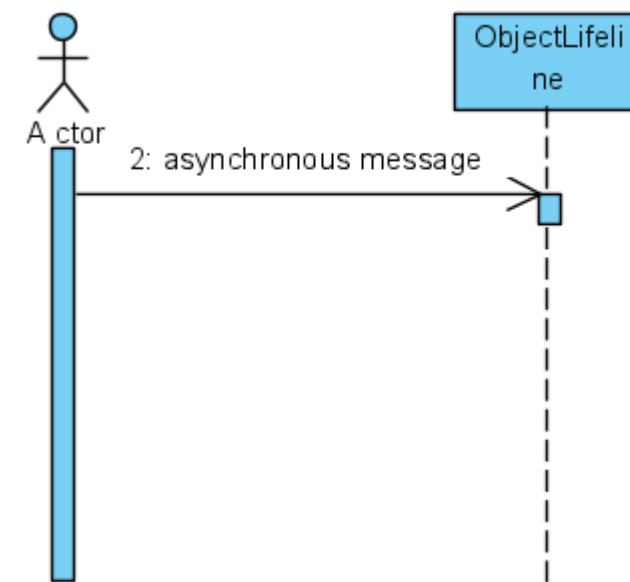
Return message [6]

- A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller. Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.



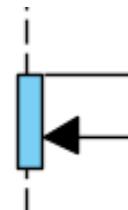
Asynchronous message [6]

- An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow.



Synchronous message of object to itself

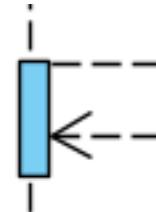
- Self message or reflexive message or message of object to itself – it is message which represent when object send a message to itself. Message arrow starts and ends at the same lifeline.
- Synchronous message of object to itself is used when the object waits for the end of the message and the feedback, before carrying on with another message.



1: synchronous message of object to itself

Return message of object to itself

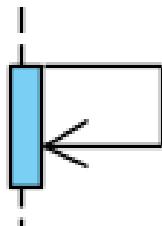
- A return message of object to itself is used to indicate that the message receiver (in this case the caller) is done processing the message and is returning control over to the message caller.
- Briefly, it is confirmation that the caller can carry on with another message (after using the synchronous message of object to itself).



4: return message of object to itself

Asynchronous message of object to itself

- An asynchronous message of object to itself is used when the message caller does not wait to process the message and return, before sending other messages to other objects within the system.



5: asynchronous message of object to itself

Example 9:

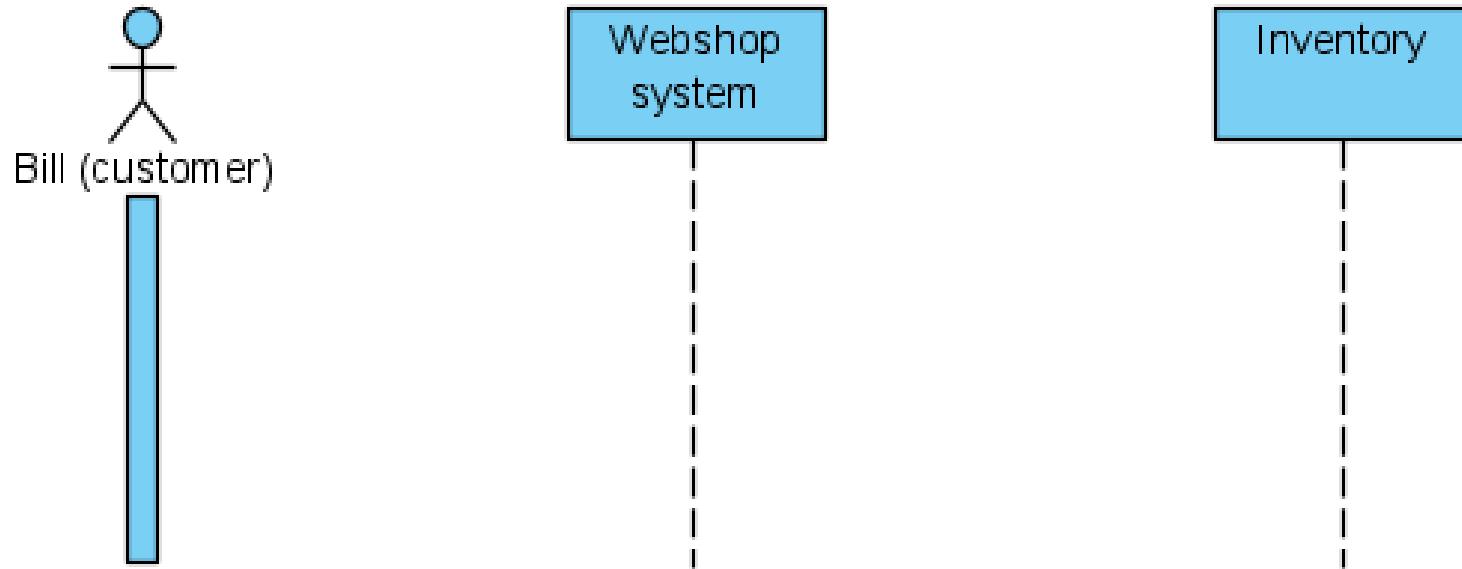
- Task

Bill (customer) wants to place an order through an online web shopping system. First, Bill presses the Order button and creates an order. The system then opens a new window (Order Window - System). For each item, Bill needs to add an item in the order.

When Bill presses the Add Item button, the system cheques the availability of the product in the inventory. Also, the inventory system sends a feedback (return message) to the web shopping system that the item is available.

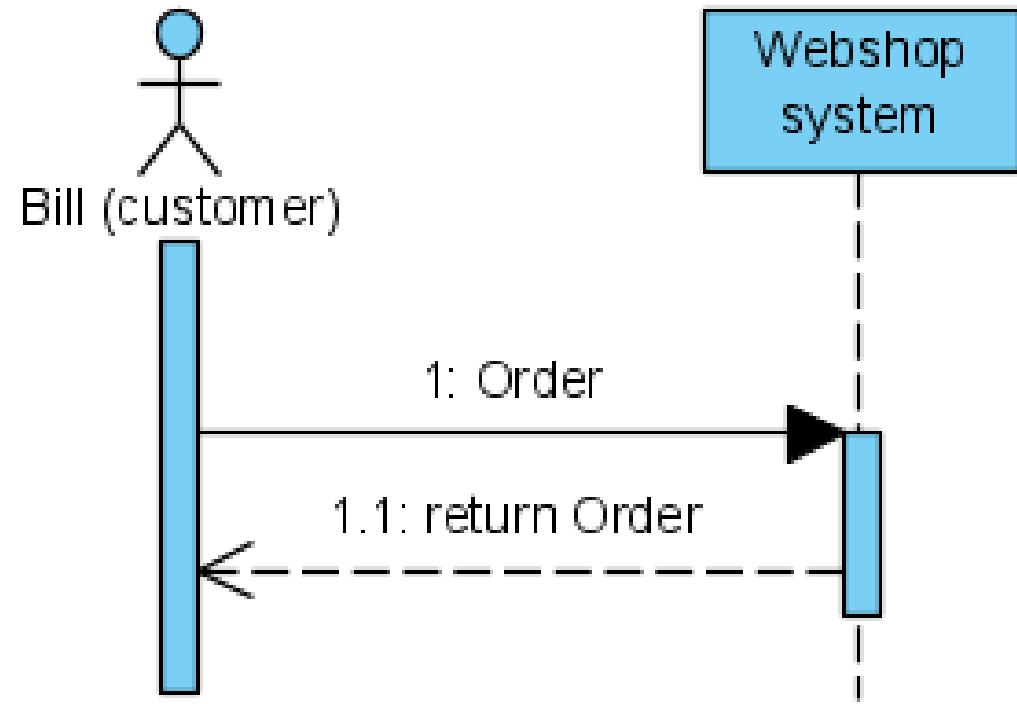
After that, the system adds the item to the order and waits for a confirmation message from its own function. At the end, the system sends the confirmation message for each added item to the customer.

Actor and objects lifelines

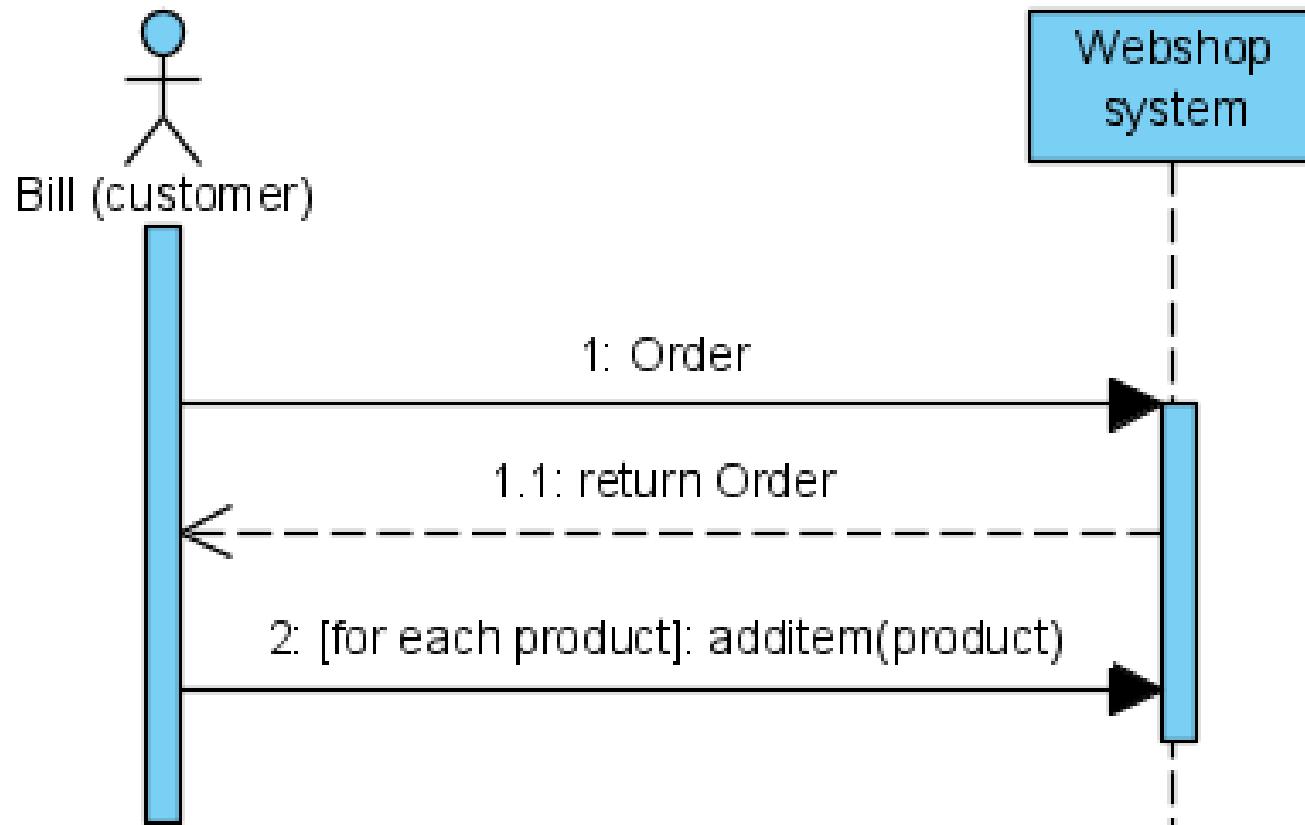


- Bill (customer) wants to place an order through an online web shopping system.

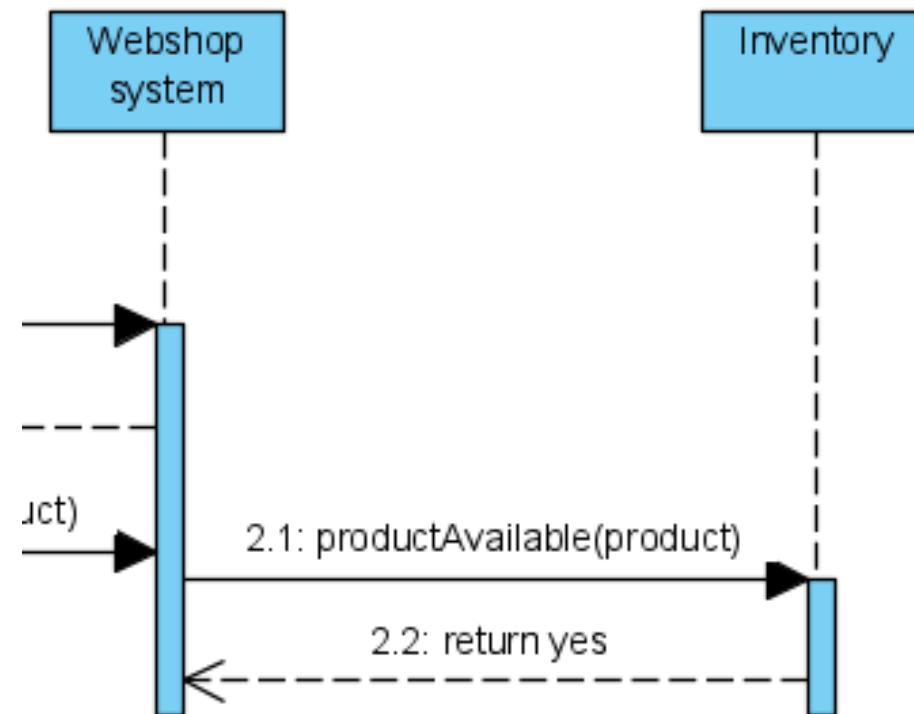
First, Bill presses the Order button and creates an order. The system then opens a new window (Order Window - System).



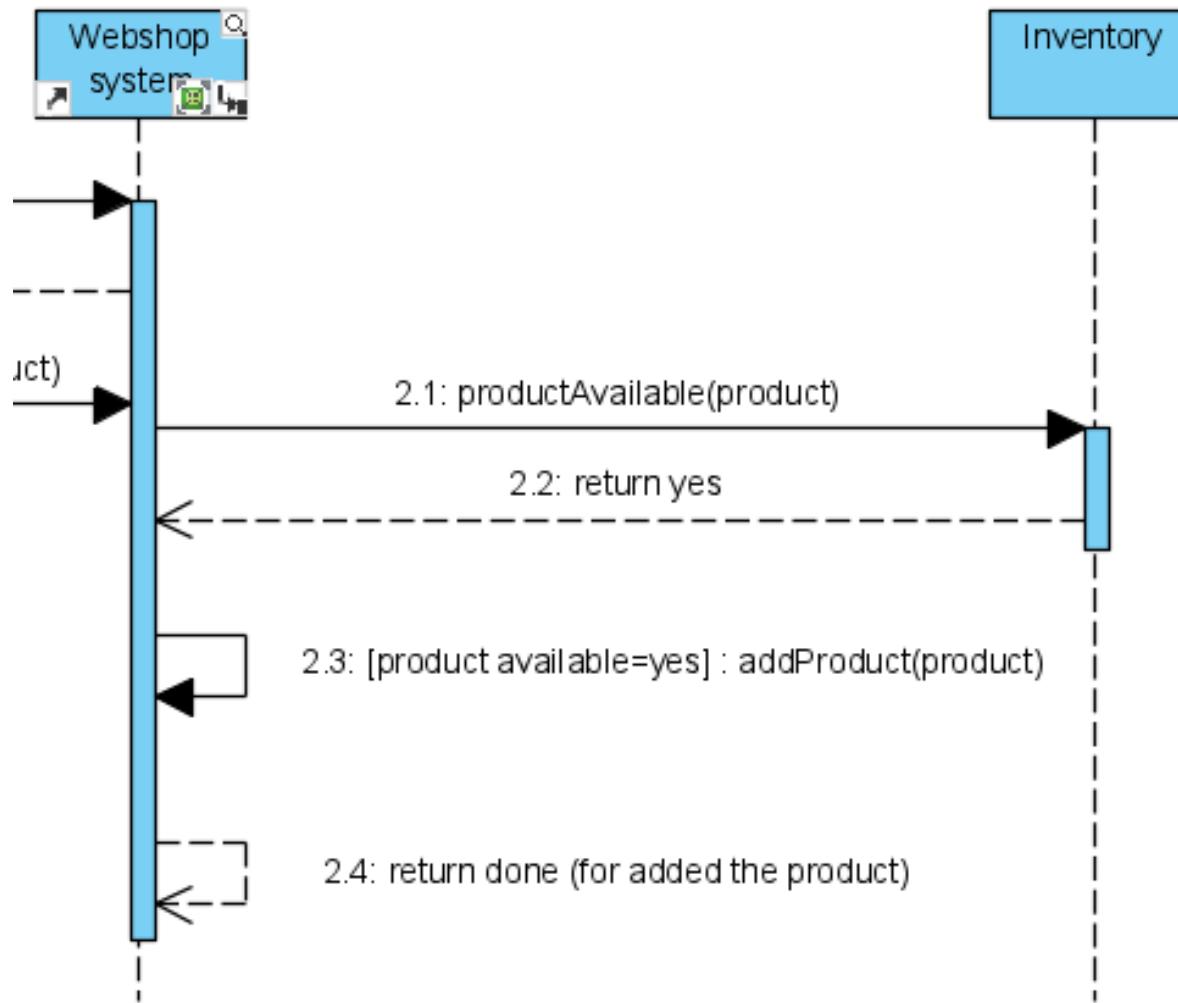
- For each item, Bill needs to add an item in the order.



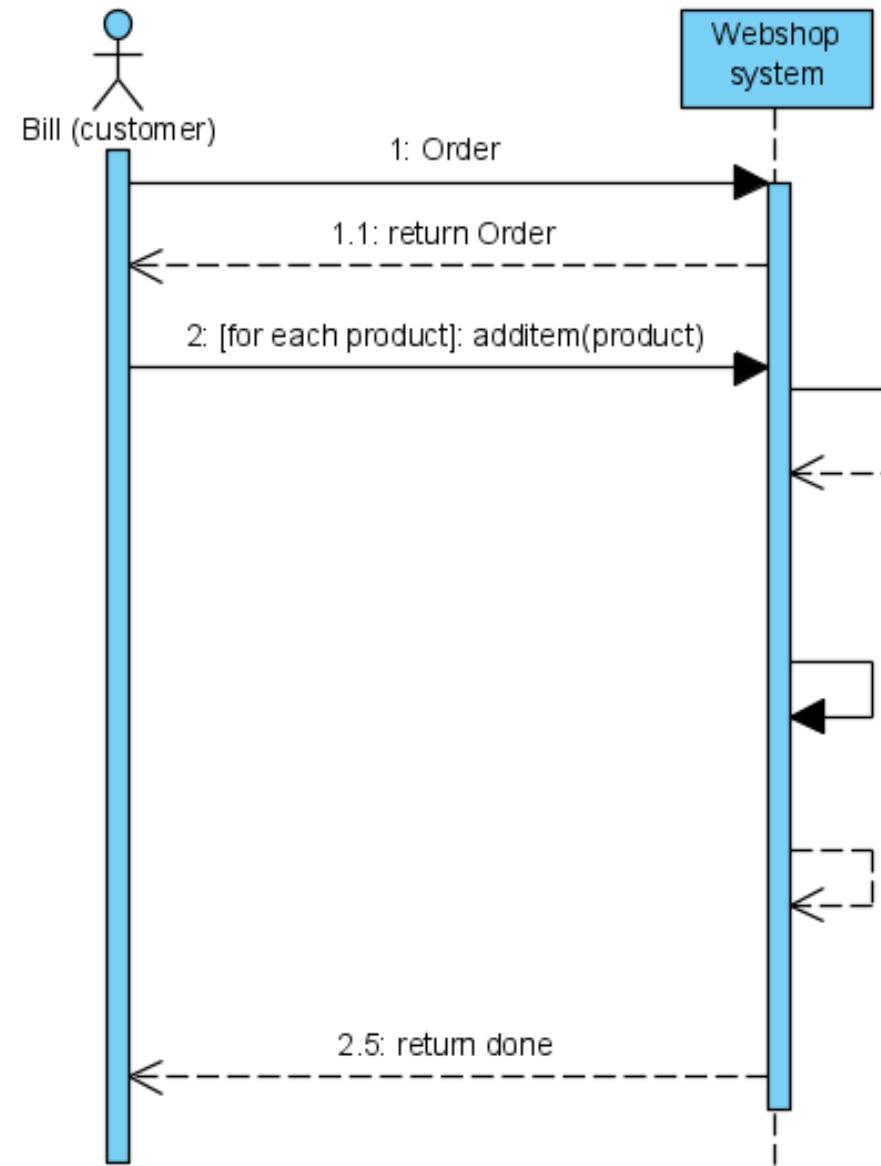
- When Bill presses the Add Item button, the system cheques the availability of the product in the inventory. Also, the inventory system sends a feedback (return message) to the web shopping system that the item is available.

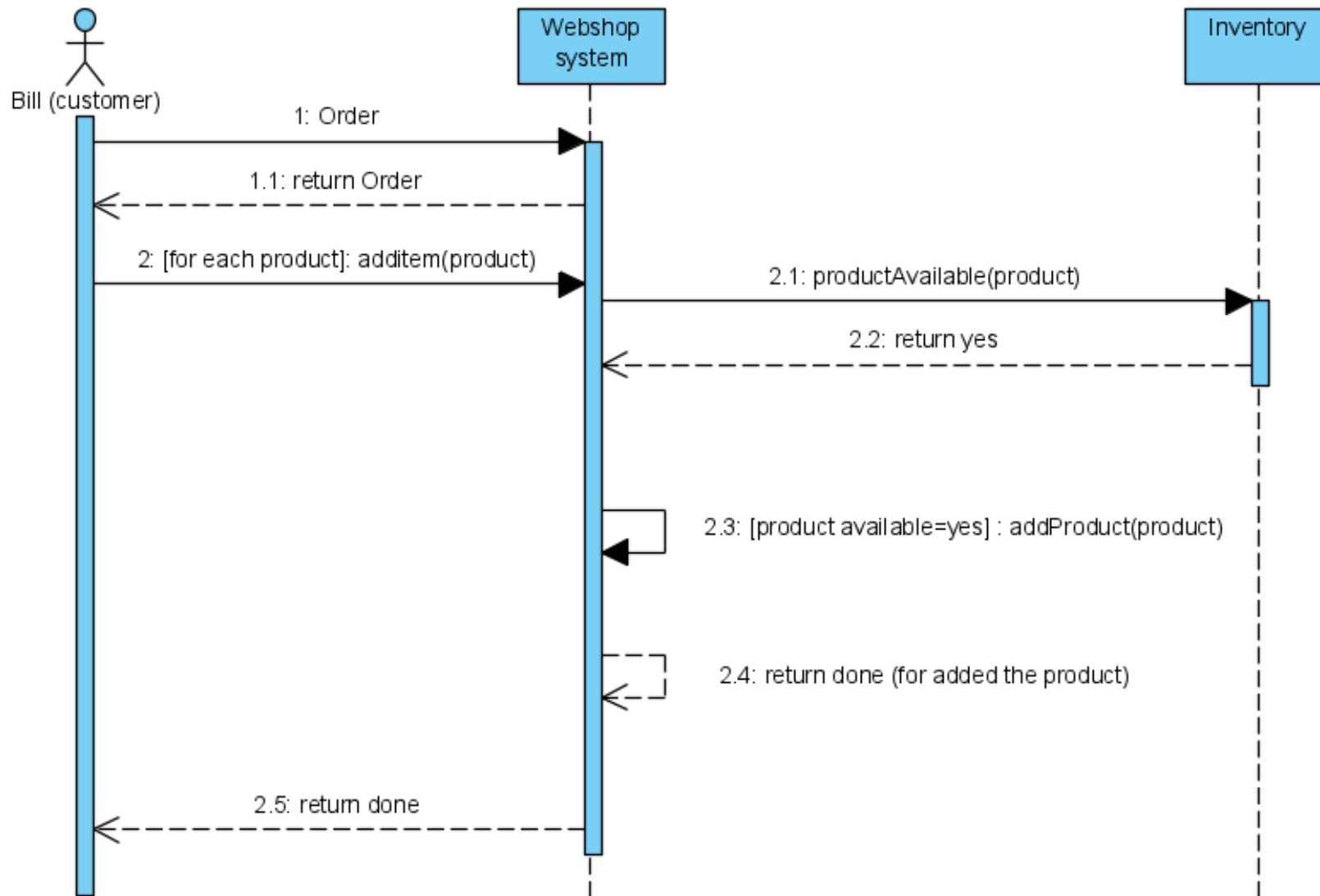


- After that, the system adds the item to the order and waits for a confirmation message from its own function.



- At the end, the system sends the confirmation message for each added item to the customer.





Example 10:

- Task

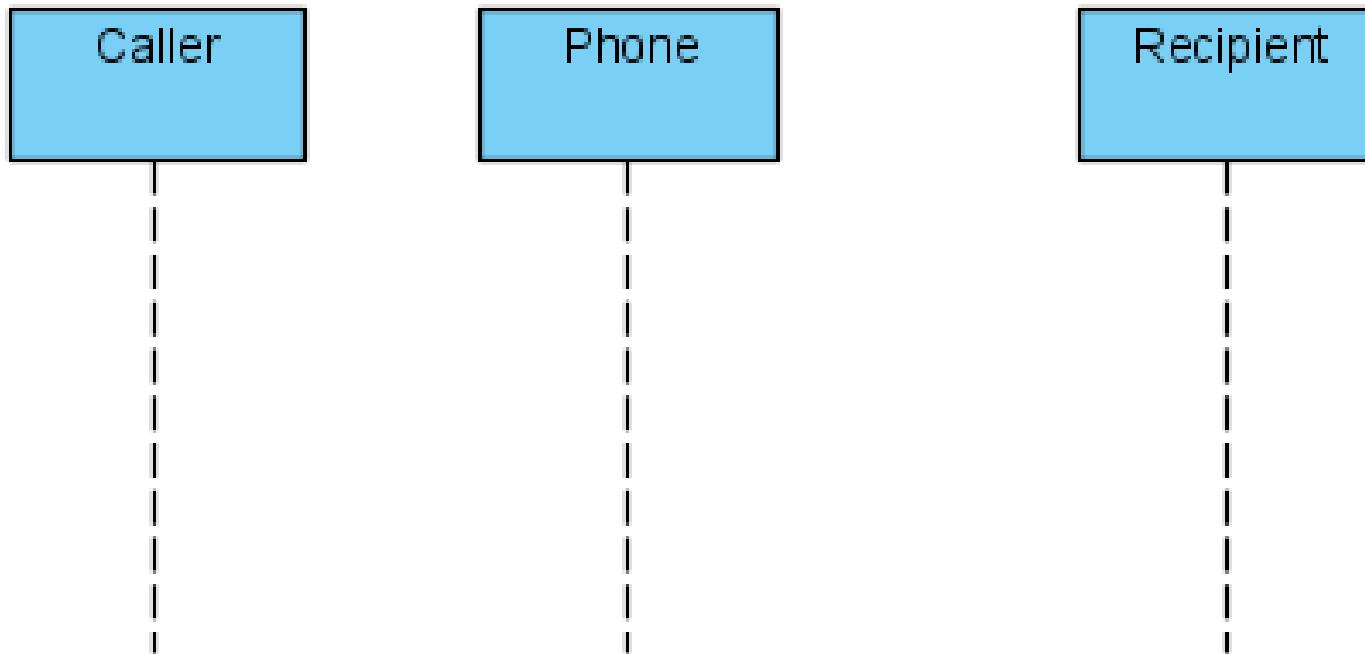
Design a sequence diagram for Phone Call procedure.

The caller picks up the phone and hears the dial tone produced by the phone. Then the caller enters (dials) the number.

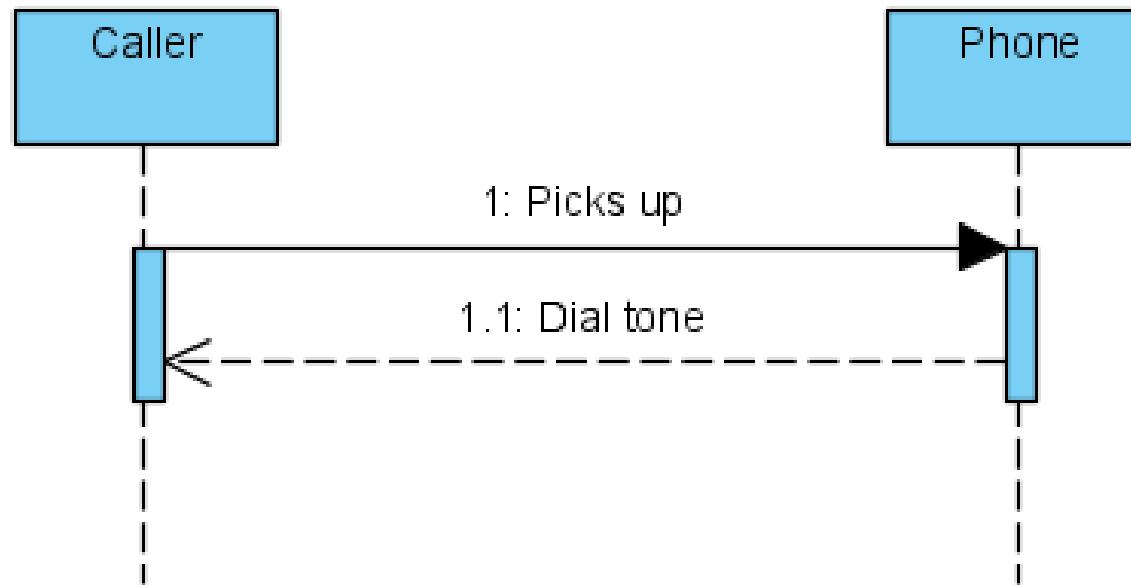
The telephone conversation is established with the receiver (the receiver's telephone rings). The caller hears the ringing notification signal. The Recipient picks up the phone and answers the call.

The caller ends the call and hears the end signal. The recipient also hears the end signal.

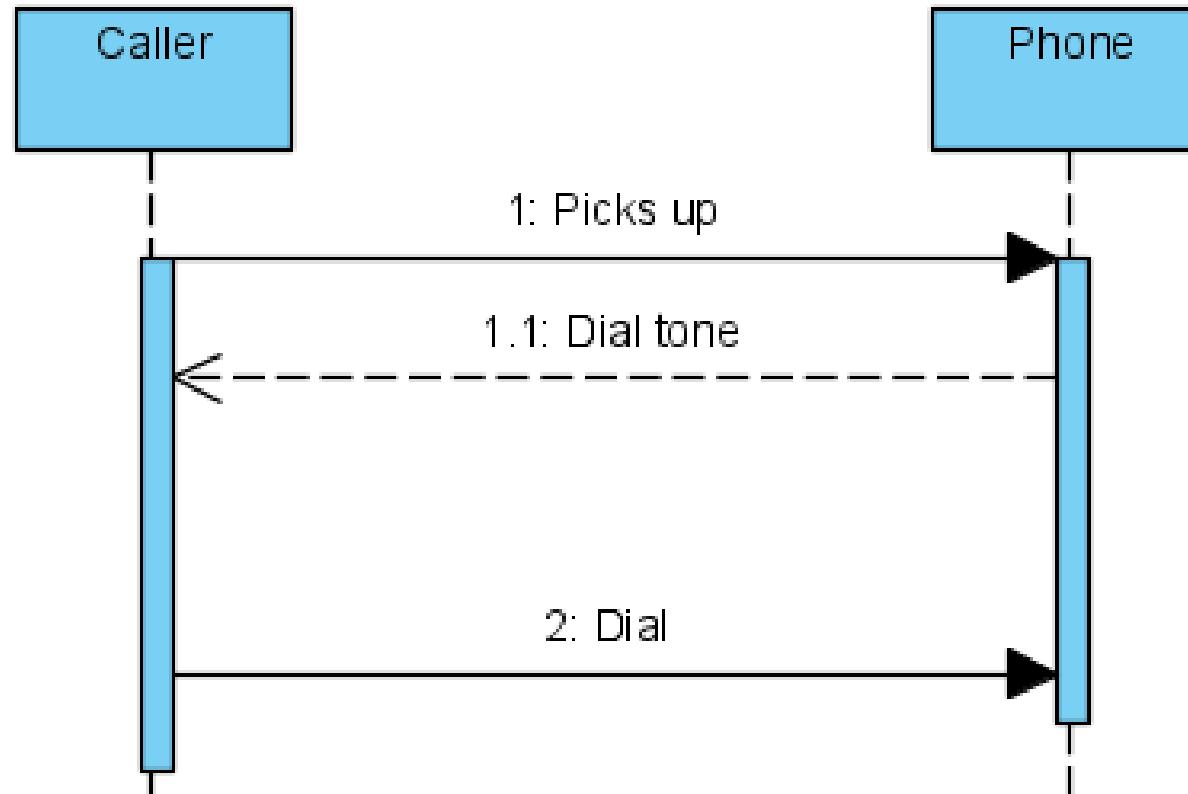
Lifelines



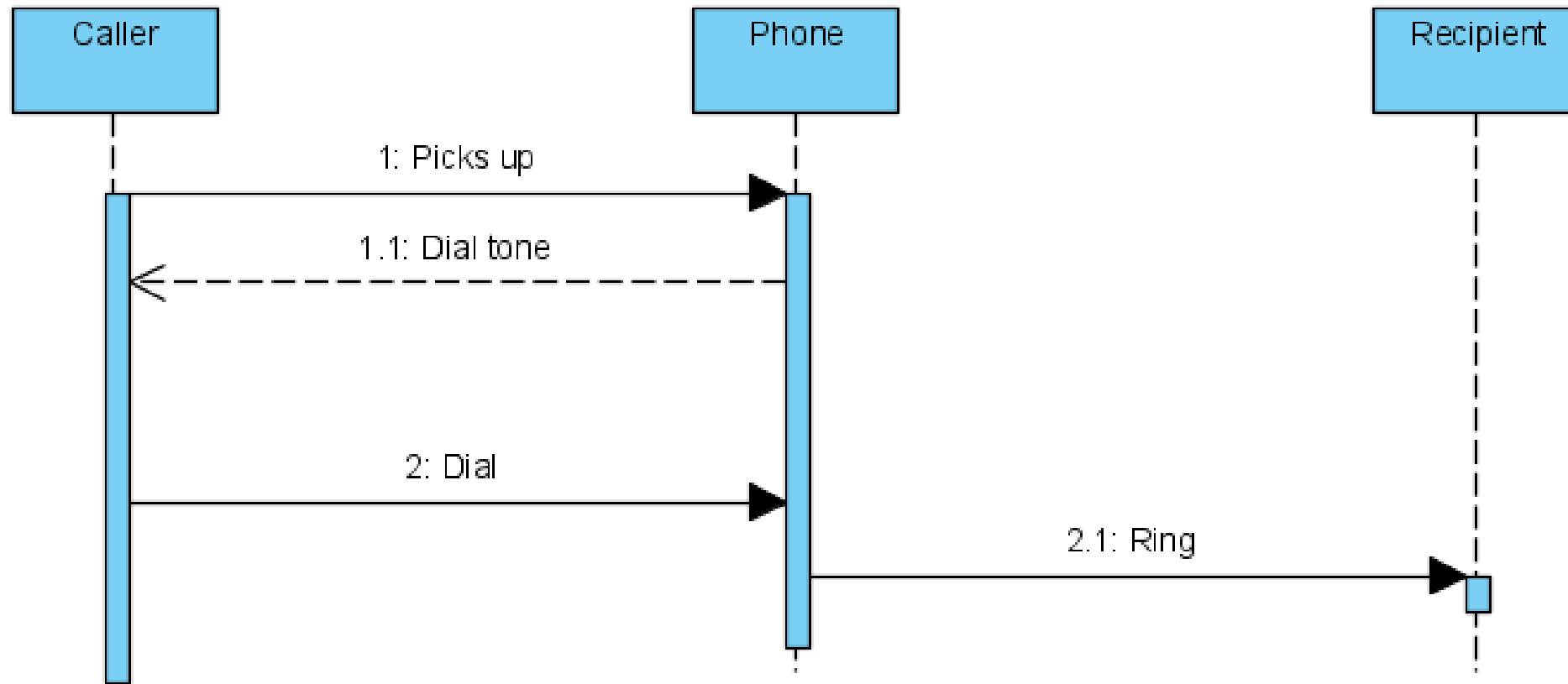
- The caller picks up the phone and hears the dial tone produced by the phone.



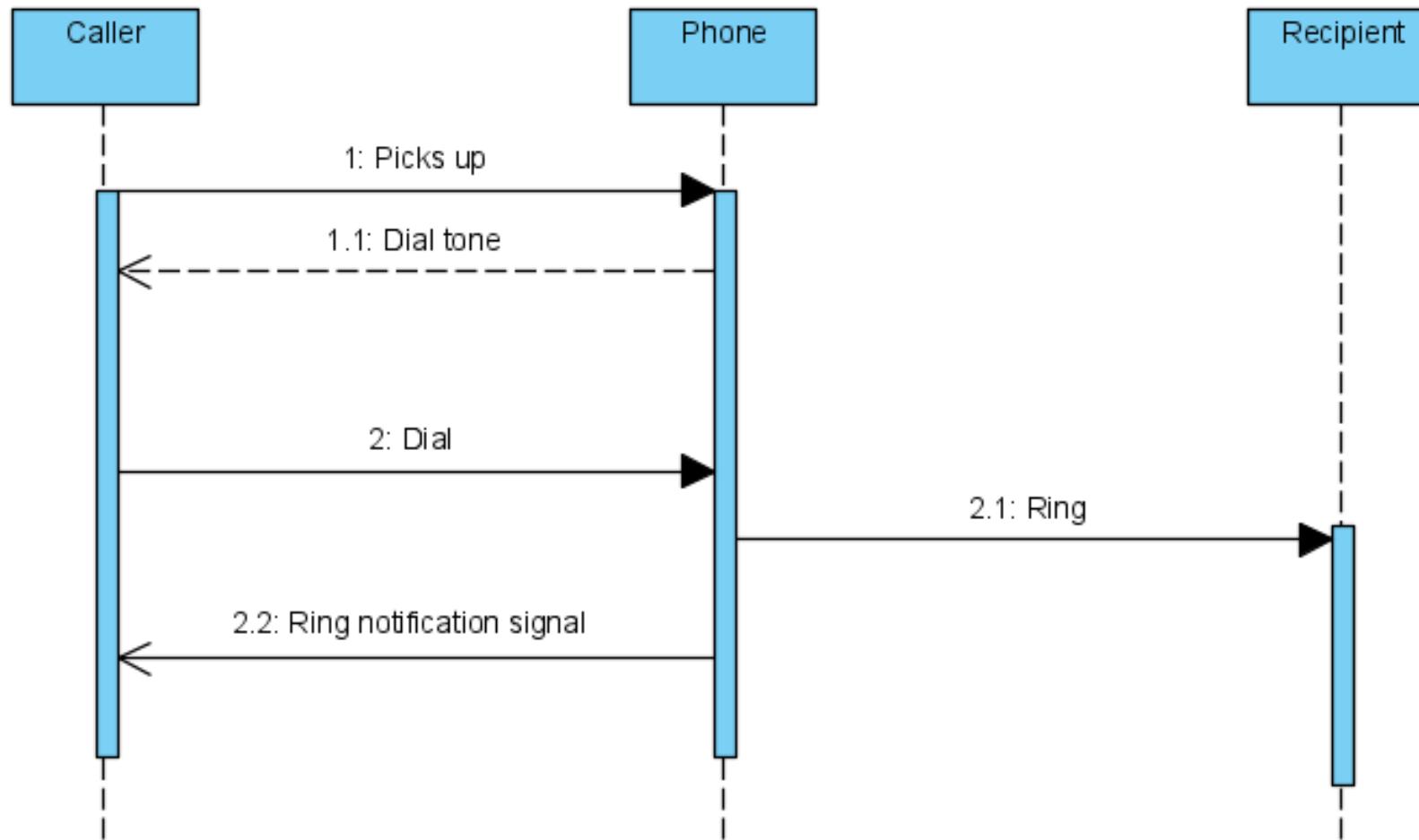
- Then the caller enters (dials) the number.



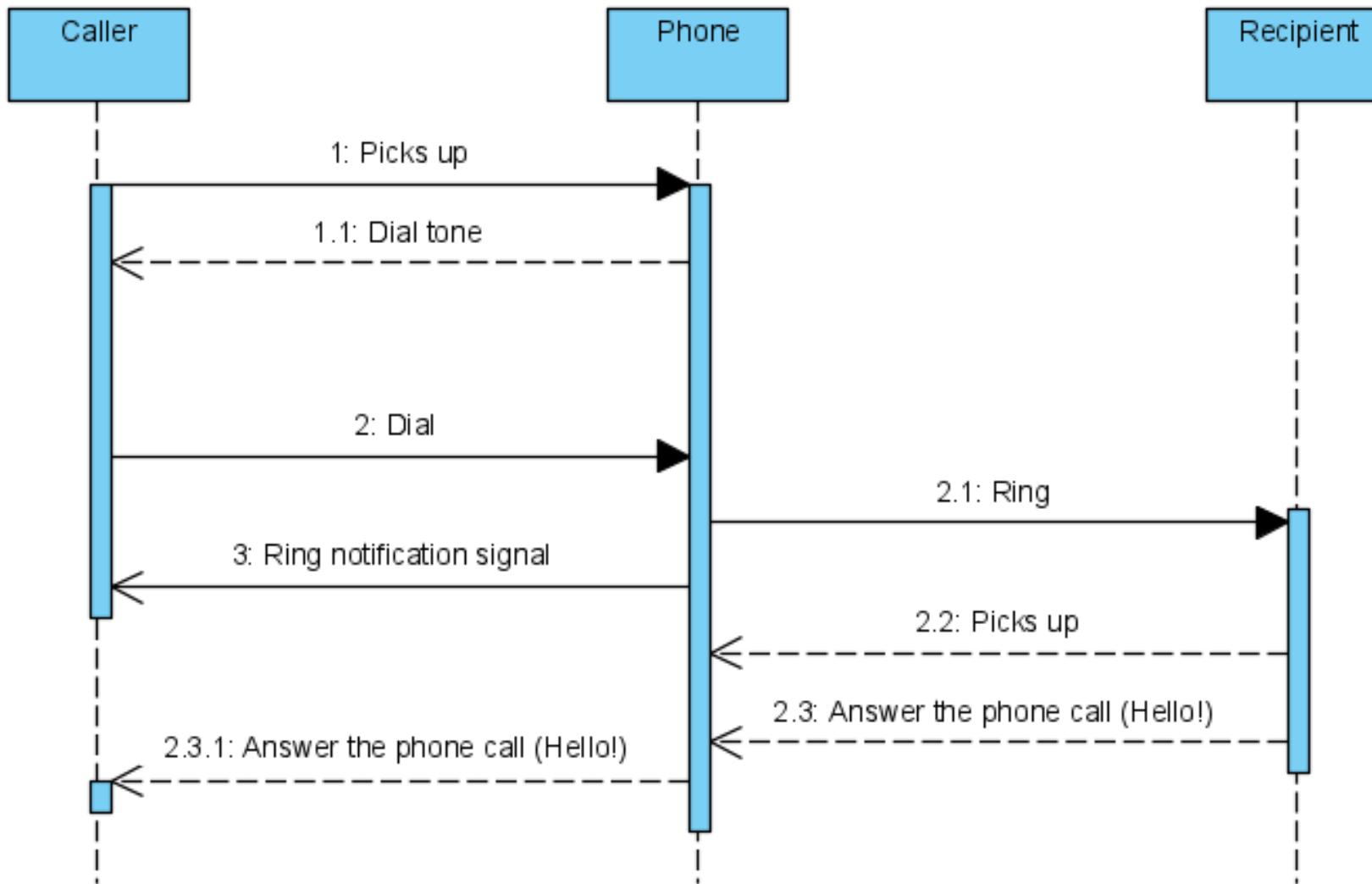
- The telephone conversation is established with the receiver (the receiver's telephone rings)



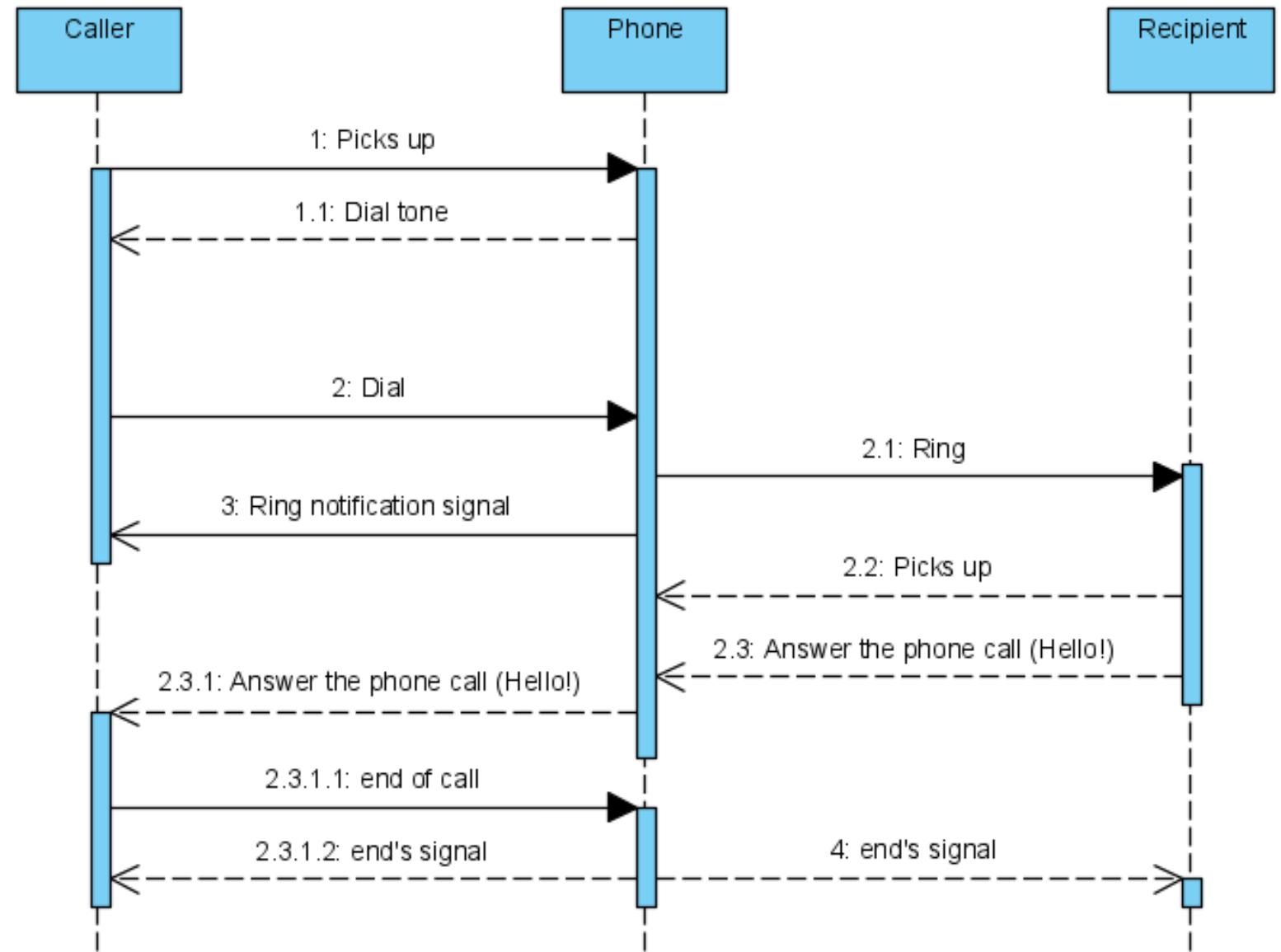
- The caller hears the ringing notification signal. - this is asynchronous message



- The Recipient picks up the phone and answers the call.



- The caller ends the call and hears the end signal. The recipient also hears the end signal.



Example 11: [7]

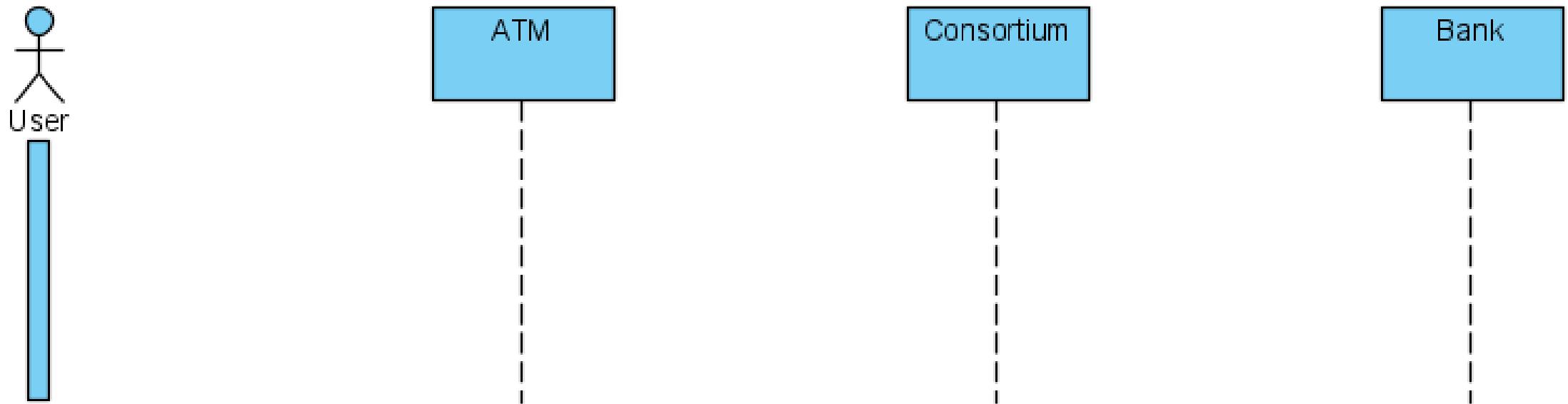
- Task

Design sequence diagram of the ATM verification card (with the rejection card at the end).

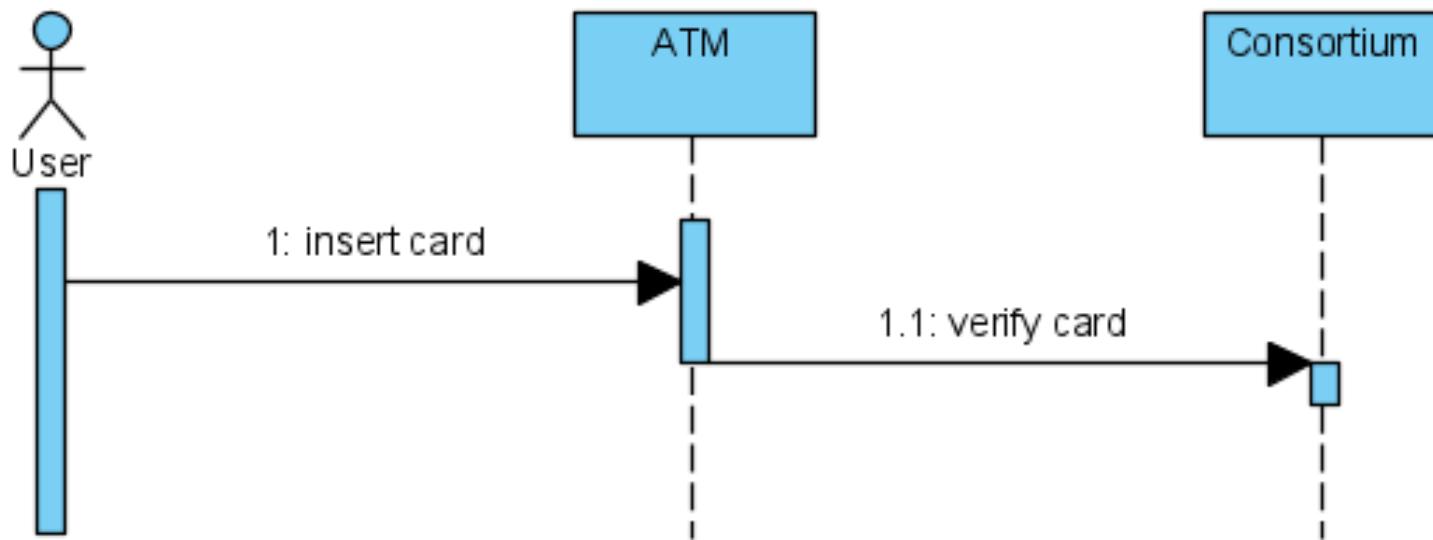
The user inserts the card into ATM, ATM sends a message to the consortium to verify the card. The consortium verifies the account in the bank. Also, the consortium system sends the response back to ATM.

The ATM sends a message back to the user (error message).

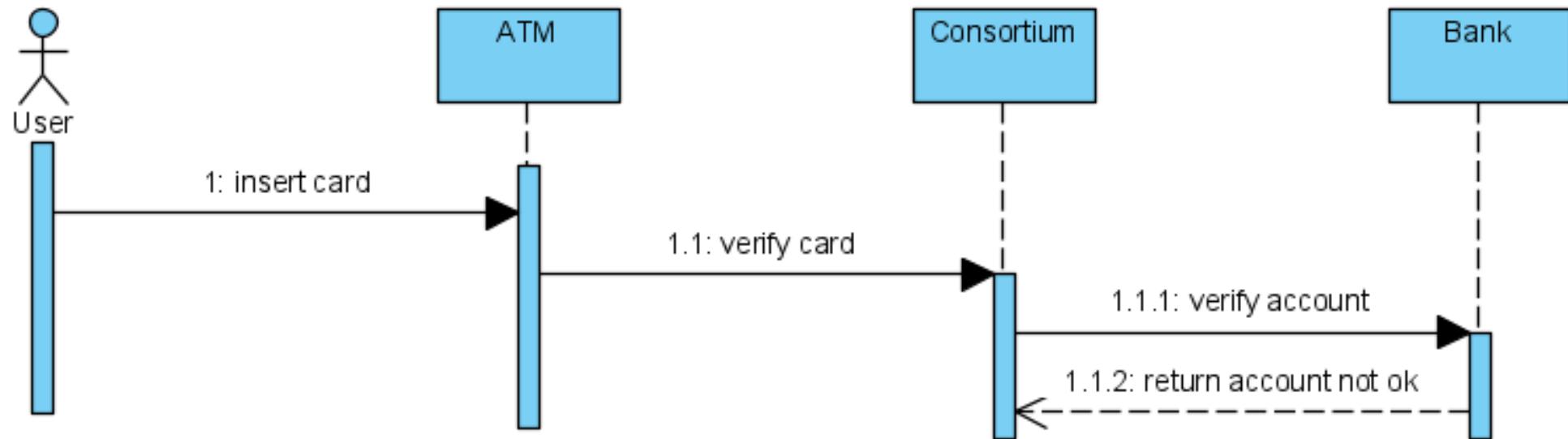
Actor and objects lifelines



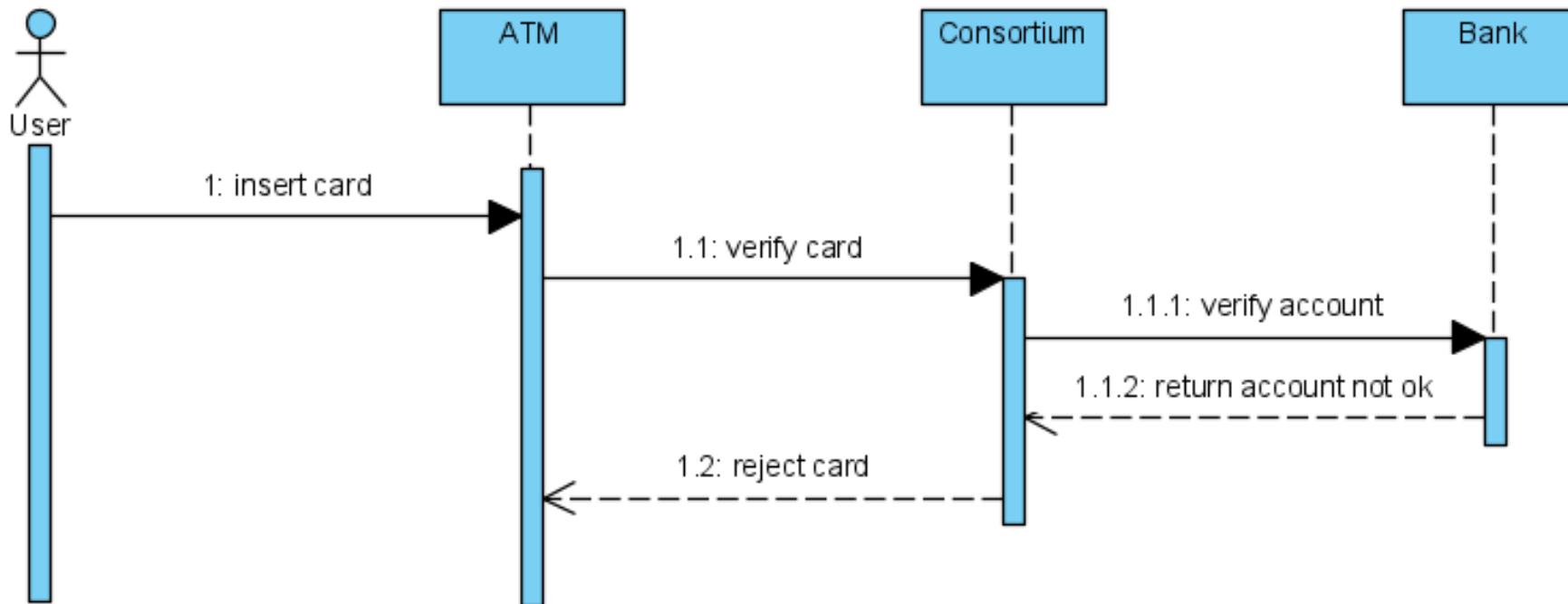
- The user inserts the card into ATM, ATM sends a message to the consortium to verify the card.



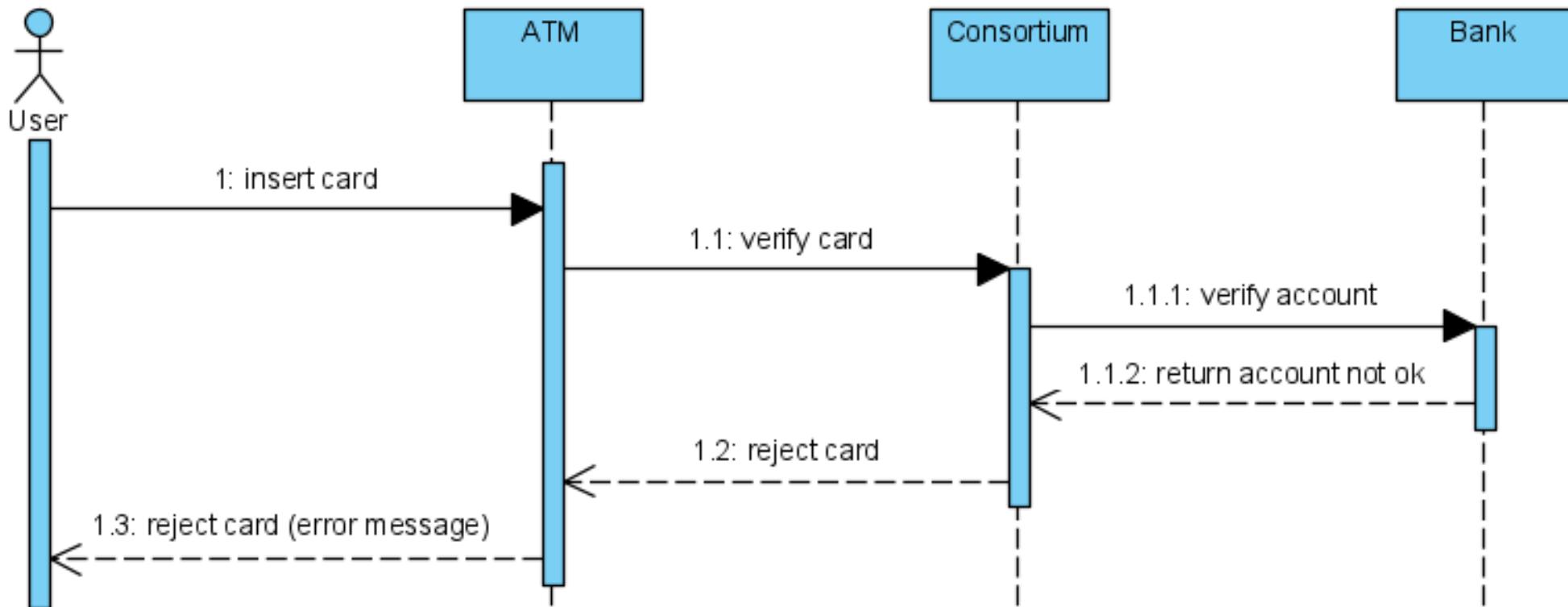
- The consortium verifies the account in the bank.



- Also, the consortium system sends the response back to ATM.



- The ATM sends a message back to the user (error message).



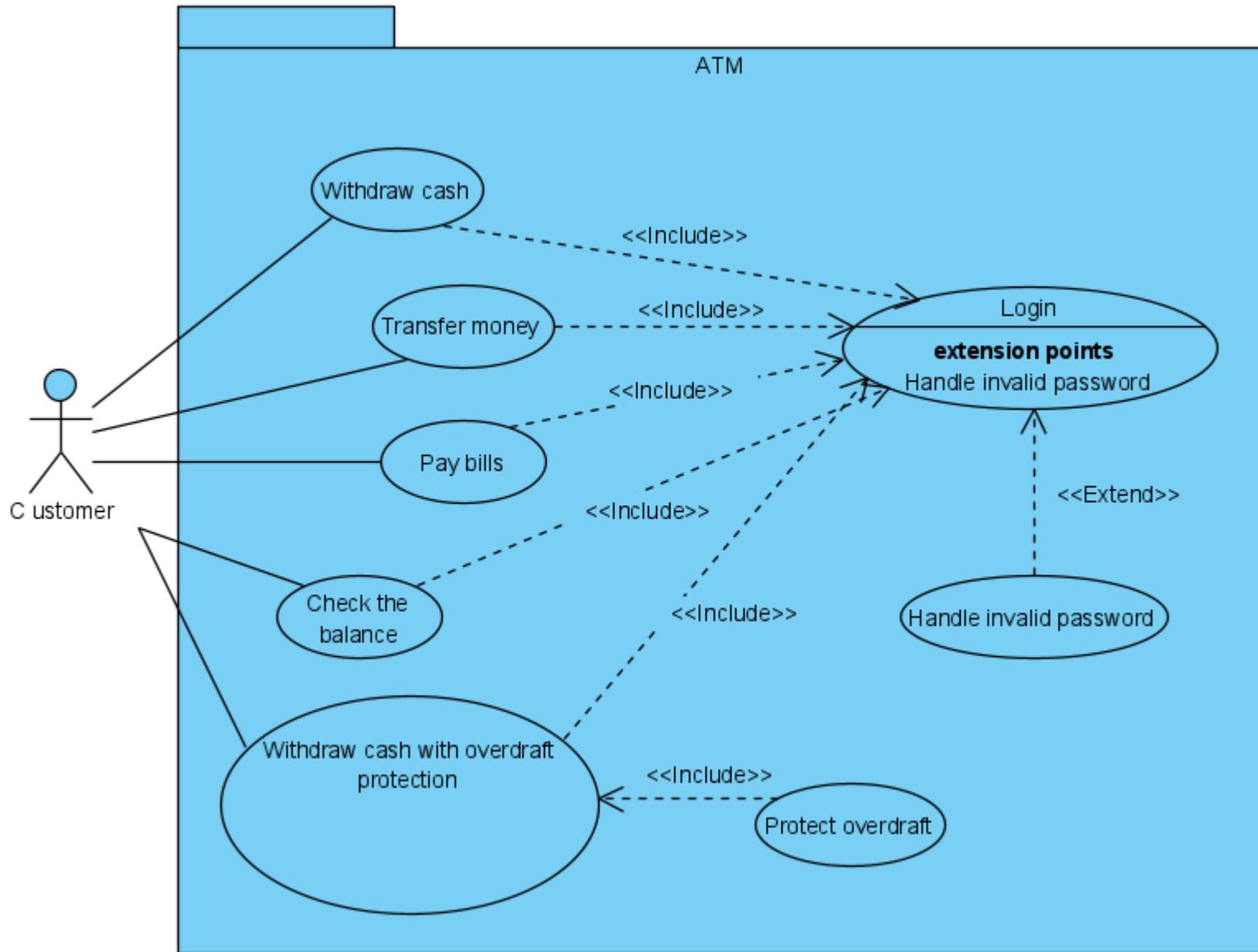
Example 12 : the same as Example 3 in chapter Use case diagrams

- Task

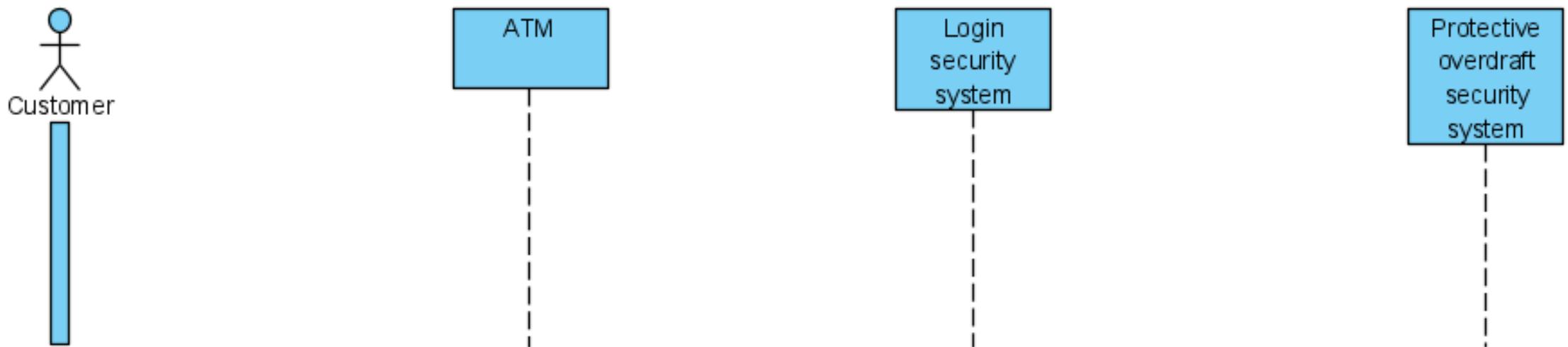
Model the ATM, given by the following description, using the UML use case diagram.

The customer needs money to pay a bill and goes to ATM. Using ATM, the customer can withdraw cash, transfer money, and pay bills. The customer can also check their account balance. All of these functions require login (the correct password). Since the customer can insert invalid password, the ATM login feature has been enhanced to handle invalid passwords.

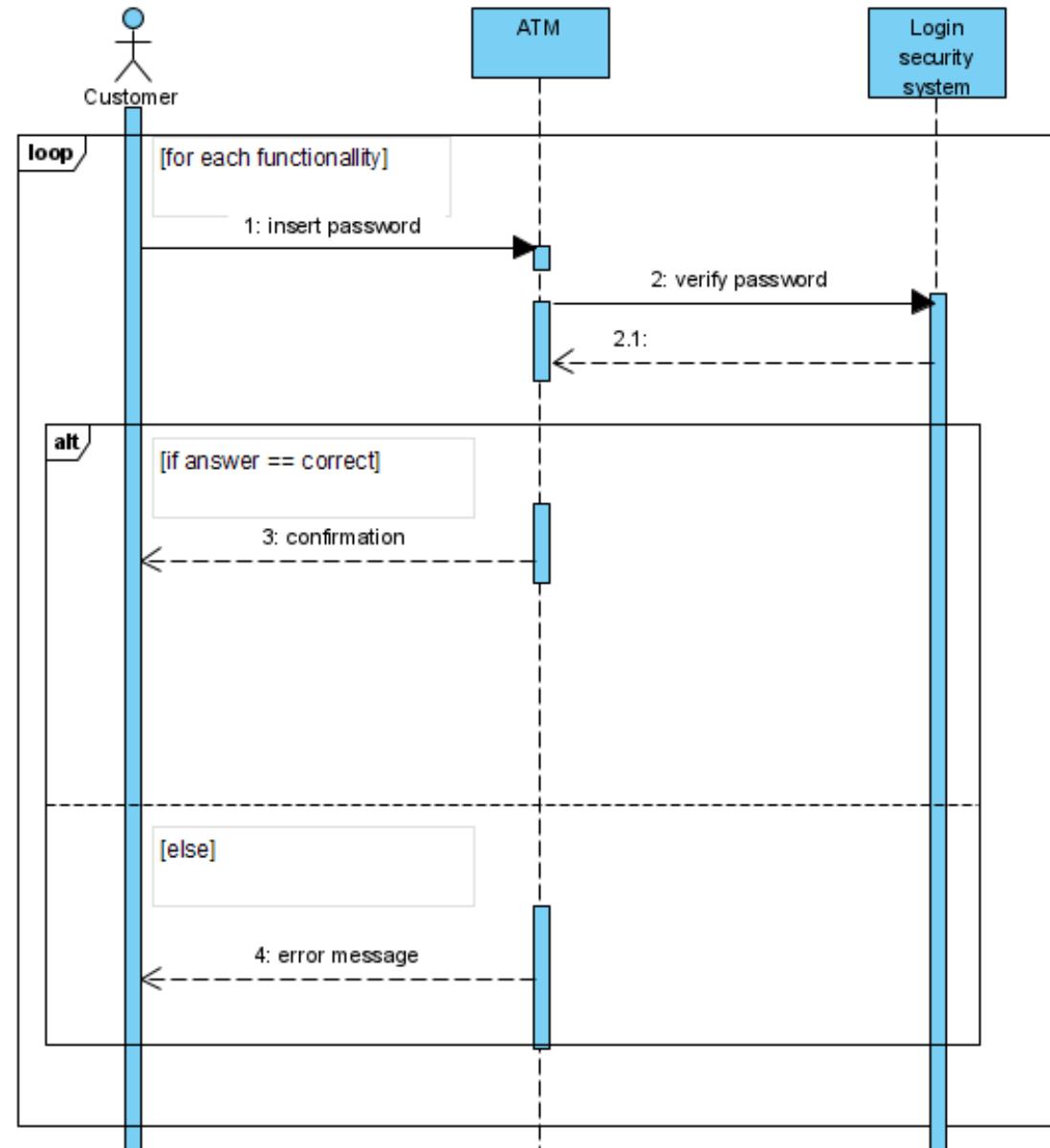
The customer can withdraw cash with an overdraft protection feature.



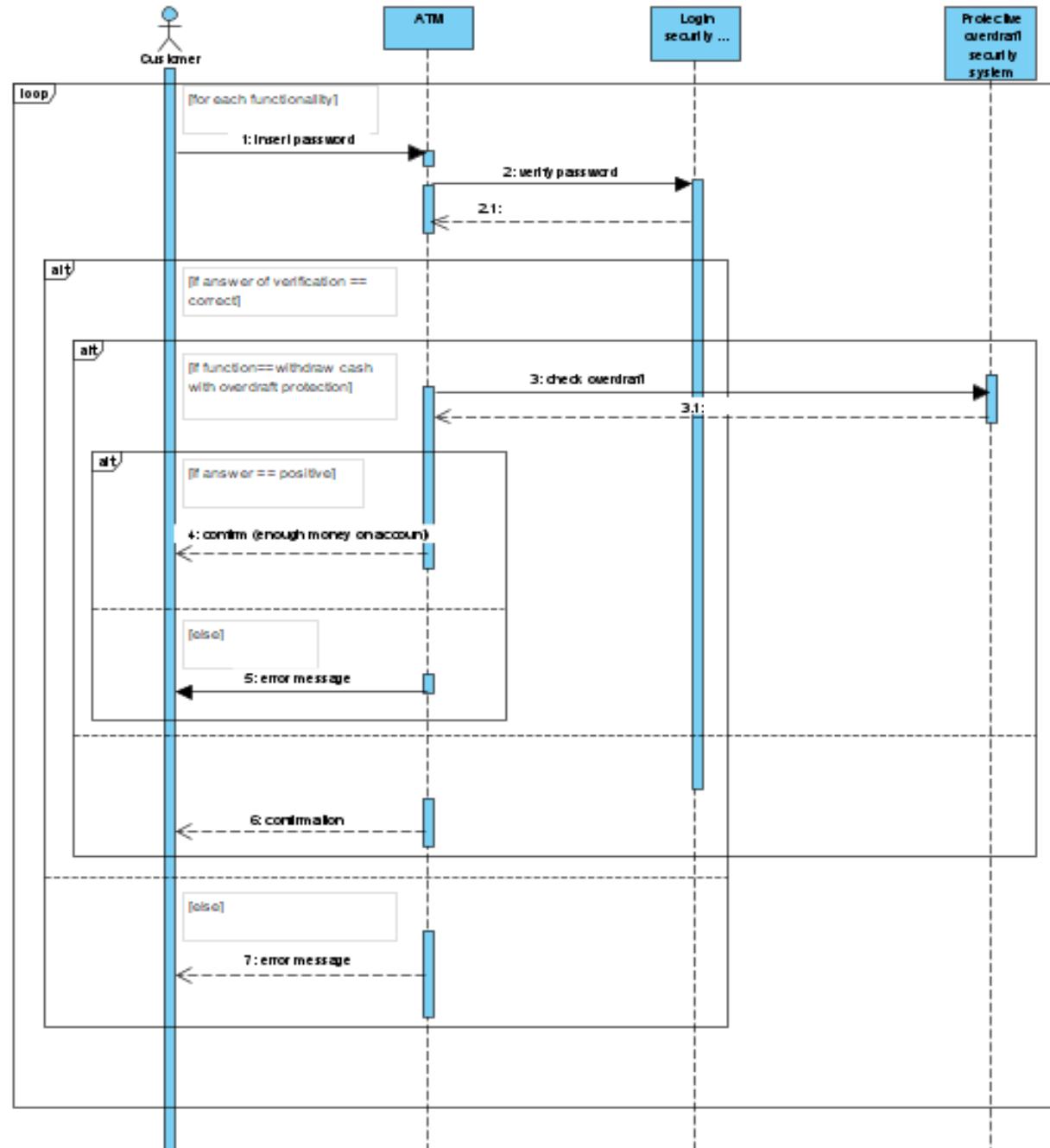
Actor and objects lifeline



The customer needs money to pay a bill and goes to ATM. Using ATM, the customer can withdraw cash, transfer money, and pay bills. The customer can also check their account balance. All of these functions require login (the correct password). Since the customer can insert invalid password, the ATM login feature has been enhanced to handle invalid passwords.



The customer can withdraw cash with an overdraft protection feature.



Use case description

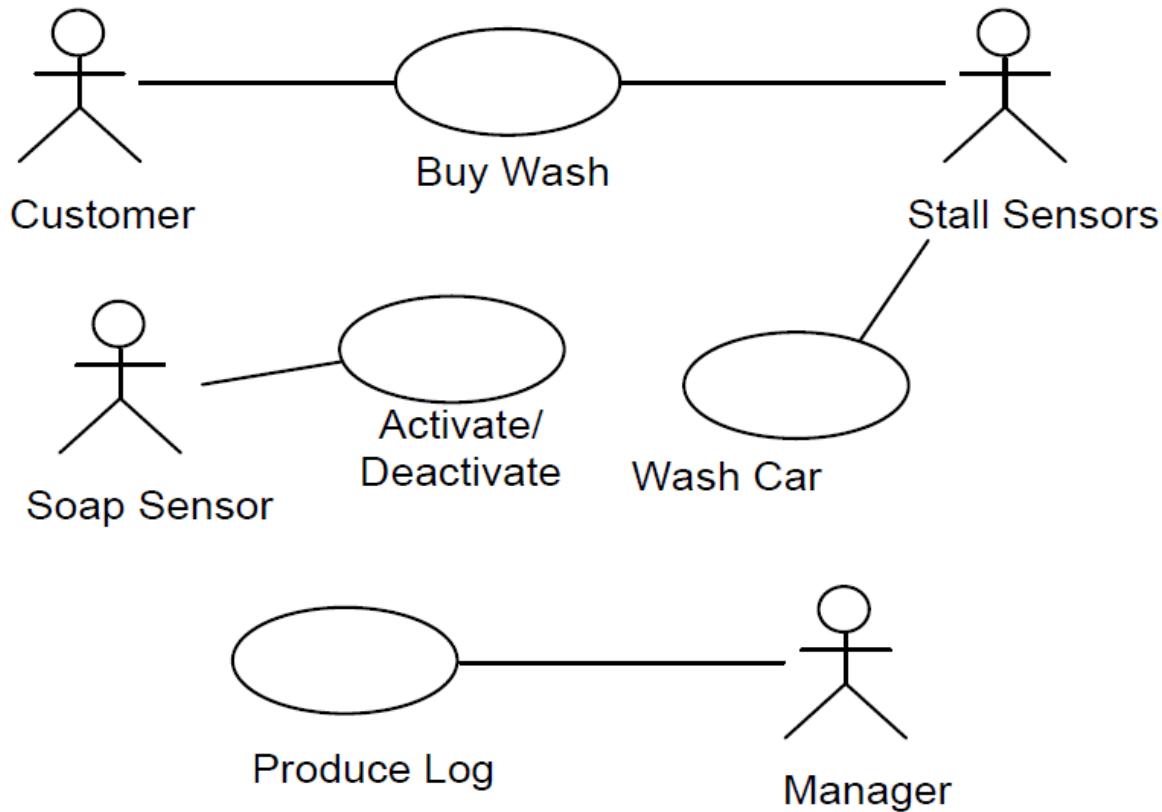
- Use case description is specification of interaction between SW product and actor(s).
- Provide information of:
 - Activities of actor and activities of product,
 - All possible interaction courses

Use case description contents

- 1. Use case name
- 2. Actors
 - List of all the actors (people, external systems...)
- 3. Requirements
 - Requirements related to the use-case.
- 4. Trigger
 - The event that starts the activity (and preconditions)
- 5. Basic (main) flow
 - The sequence of user actions and system responses that will take place during execution of a use case and leads to accomplishing the goal.
- 6. Post conditions
 - Expected outcome of the use case with the state of the system at the conclusion of the use case execution.

- 7. Alternative flows
 - List and describe other legitimate usage scenarios that can take place with the use case.
 - May start and end in anywhere related to other flows.
 - May be based on other alternative flows.
- 8. Other
 - Use case comments and links to other use cases.

Use case diagram example – A use case diagram for a car wash



Use case diagram description (1)- example

- Name: Activate/Deactivate
- Actor: Soap sensor
- Requirements:
 - Customers — Need their cars washed with soap, and want a complete wash
 - Operations — Want the carwash to operate without constant attention
- Trigger: One minute has passed since the last time the soap sensor was checked.
- Basic flow:
 - 1.The carwash queries the soap sensor.
 - 2.The soap sensor indicates that there is soap.
 - 3.If the carwash is active, it continues its operation and the use case ends.
- Postconditions:
 - The carwash is active if and only if the soap sensor indicates that there is soap.
 - No wash currently in progress is interrupted.

Use case diagram description (2)- example

- Alternative flows:
- 1a: The carwash is unable to query the soap sensor:
 - 1a1. The controller logs the problem and the use case ends.
- 2a: The soap sensor indicates that there is no more soap:
 - 2a1. If the carwash is inactive, the use case ends.
 - 2a2. If the carwash is active, the controller displays an out-of-order message and becomes inactive; the use case ends.
- 2b: The soap sensor does not respond:
 - 2b1. The controller logs the problem.
 - 2b2. If the carwash is inactive, the use case ends.
 - 2b3. If the carwash is active, the controller displays an out-of-order message and becomes inactive; the use case ends.
- 2a2, 2b3: A wash is in progress:
 - 2a21. The carwash completes the current wash, then displays an out-of-order message; the use case ends.
- 3a: The carwash is inactive:
 - 3a1. The carwash becomes active and displays a ready message.

Homework assignment

- Agreement on use case diagram and user story + use case descriptions (each member - one use case description).
- Within the group, you must agree on the choice of use case diagram and user story.
- If you submit the same user stories of one of member or the same user diagrams, you will receive 0 points. You need to merge your stories and diagrams and create better versions of both.
- Each member of the group must write a one (different) use case description (individually - please emphasise which member is the author of which use case description).
- As a group, you need to upload a pdf file with the agreed user story and use case diagram, and with all use case descriptions (emphasise which author wrote which use case description).

UML state machine diagram

Definition (1)

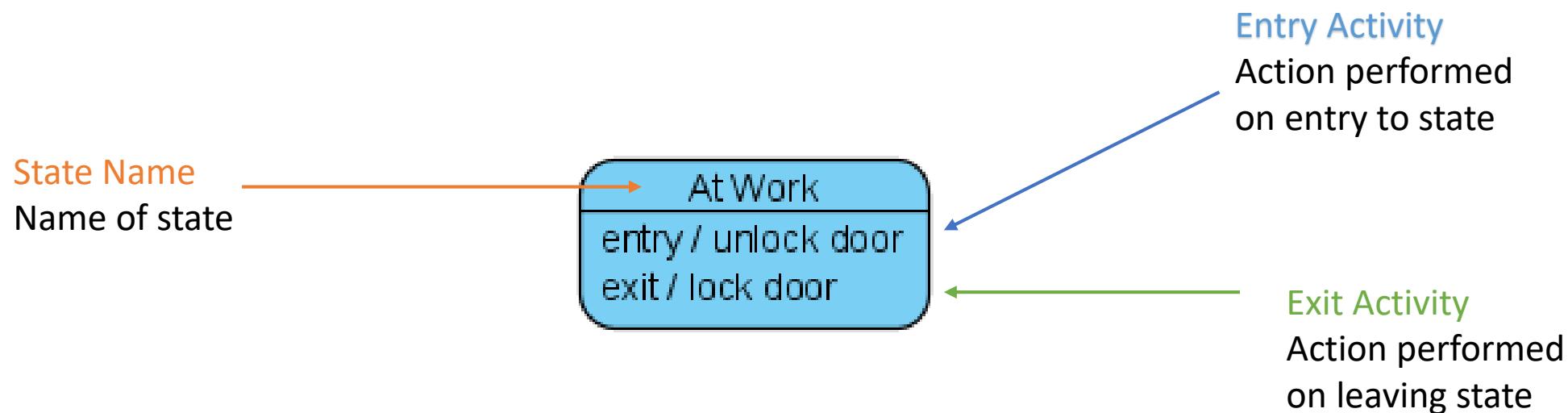
- **State machine diagram** is used to describe the different states of a component within a system.
- Some objects act differently dependent to a state in which they are in certain moment.
- An object state depends on previous activities and changes due to (external) events.
- A state can be associated with a characteristic activity, that executes at transition to this state.
- The state machine diagram depicts possible object states and transitions between the states, events that influence state changes, and activities related to states and transitions.

Definition (2) [1]

- UML State Machine Diagrams (or sometimes referred to as state diagram, state machine or state chart) show the different states of an entity.
- State machine diagrams can also show how an entity responds to various events by changing from one state to another.
- State machine diagram is a UML diagram used to model the dynamic nature of a system.

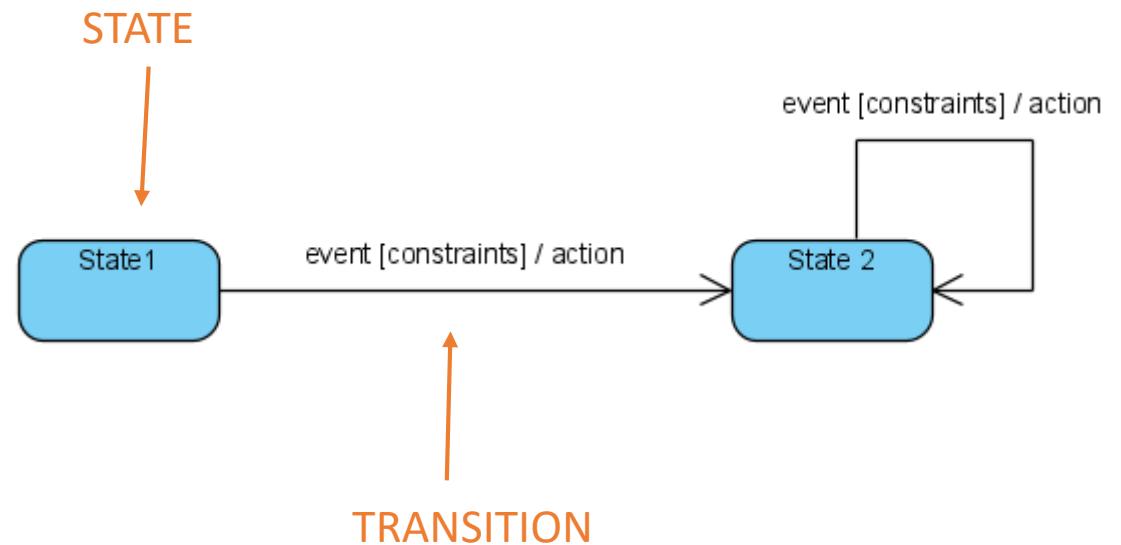
State Machine Diagram Notation

State Machine Diagram Notation [1]



State [1]

- A state is a constraint or a situation in the life cycle of an object, in which a constraint holds, the object executes an activity or waits for an event.
- A state machine diagram is a graph consisting of:
 - States (simple states or composite states)
 - State transitions connecting the states

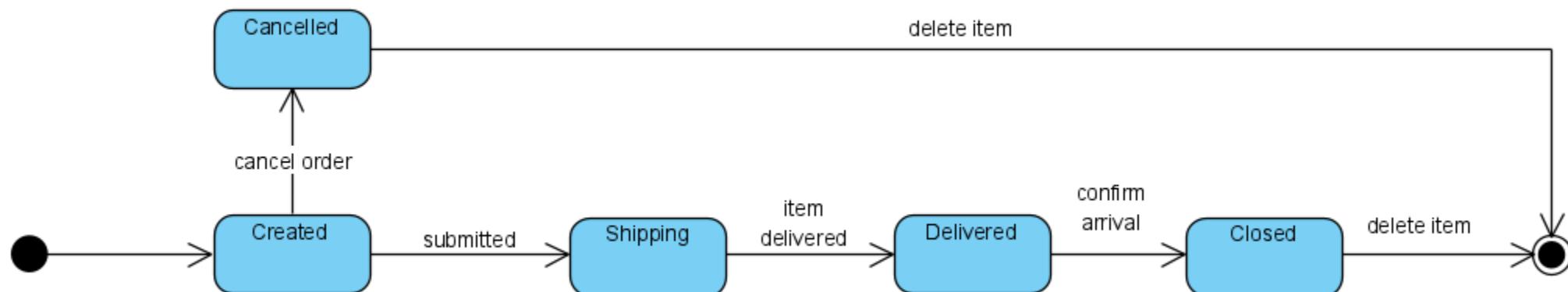


Characteristics of State [1]

- State represent the conditions of objects at certain points in time.
- Objects (or Systems) can be viewed as moving from state to state
- A point that satisfies some condition, where some particular action is being performed or where some event is waited

Initial and Final States [1]

- The **initial state** of a state machine diagram, known as an initial pseudo-state, is indicated with a solid circle. A transition from this state will show the first real state
- The **final state** of a state machine diagram is shown as concentric circles. An open loop state machine represents an object that may terminate before the system terminates, while a closed loop state machine diagram does not have a final state; if it is the case, then the object lives until the entire system terminates.



Events [1]

- An event signature is described as Event-name (comma-separated-parameter-list). Events appear in the internal transition compartment of a state or on a transition between states. An event may be one of four types:
 - Signal event - corresponding to the arrival of an asynchronous message or signal
 - Call event - corresponding to the arrival of a procedural call to an operation
 - Time event - a time event occurs after a specified time has elapsed
 - Change event - a change event occurs whenever a specified condition is met

Characteristics of Events [1]

- Represents incidents that cause objects to transition from one state to another.
- Internal or External Events trigger some activity that changes the state of the system and of some of its parts
- Events pass information, which is elaborated by Objects operations. Objects realize Events
- Design involves examining events in a state machine diagram and considering how those events will be supported by system objects

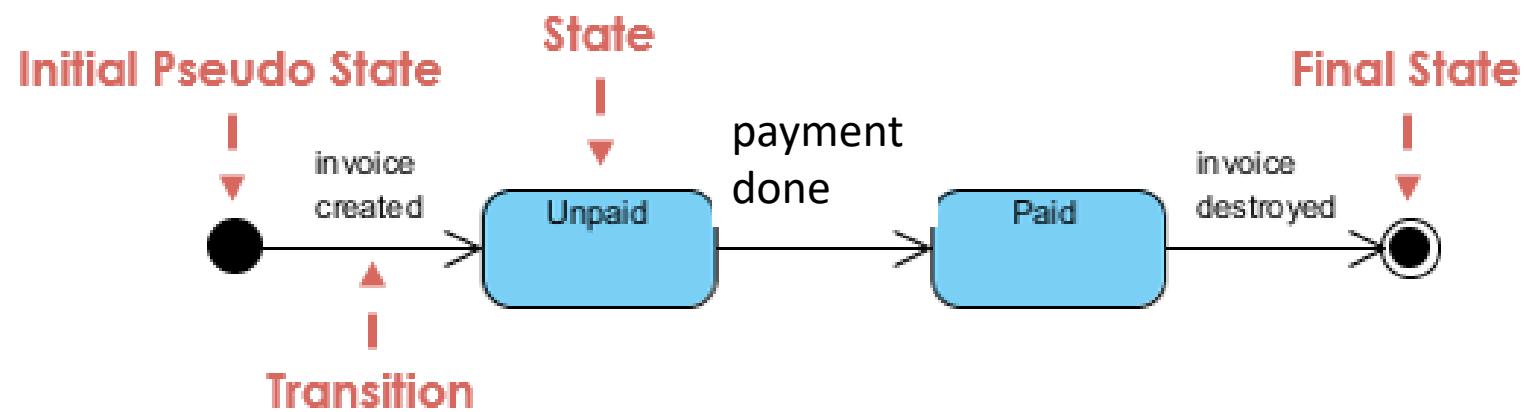
Transition [1]

- Transition lines depict the movement from one state to another. Each transition line is labeled with the **event** that causes the transition.
- Viewing a system as a set of states and transitions between states is very useful for describing complex behaviors
- Understanding state transitions is part of system analysis and design
- A Transition is the movement from one state to another state
- Transitions between states occur as follows:
 - An element is in a source state
 - An event occurs
 - The element enters a target state
- Multiple transitions occur either when different events result in a state terminating or when there are guard conditions on the transitions

Actions and Activity [1]

- Action is an executable atomic computation, which includes operation calls, the creation or destruction of another object, or the sending of a signal to an object. An action is associated with transitions and during which an action is not interruptible
- Activity is associated with states, which is a non-atomic or ongoing computation. Activity may run to completion or continue indefinitely. An Activity will be terminated by an event that causes a transition from the state in which the activity is defined- e.g., entry, exit

Simple State Machine Diagram Notation [1]



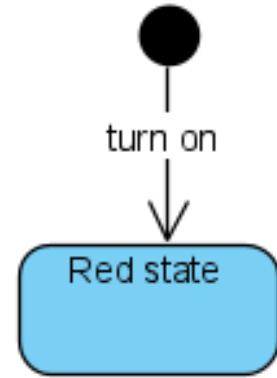
Entry and Exit Actions

- Entry and Exit actions specified in the state. It must be true for every entry / exit occurrence. If not, then you must use actions on the individual transition arcs
- **Entry Action** executed on entry into state with the **notation: Entry / action**
- **Exit Action** executed on exit from state with the **notation: Exit / action**

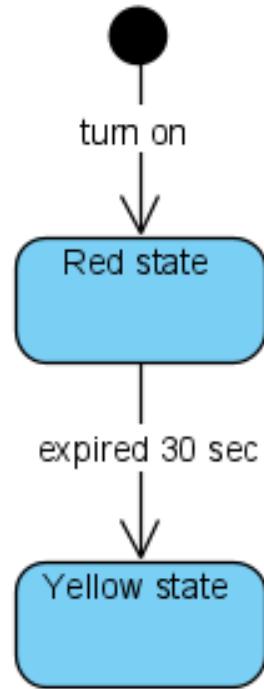
Example 1

- Model the state machine diagram of the traffic light. Start by turning on the current of the traffic light. The red light comes on first. The transition to yellow occurs after the 30 seconds have elapsed. After 1 second of the yellow signal, the green signal begins. The green signal goes out after 20 seconds and goes to yellow signal, which takes 5 seconds before lighting the red signal. After that the process repeats continuously.

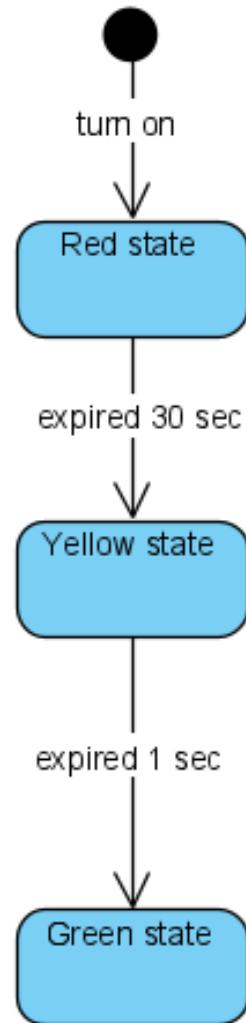
- Model the state machine diagram of the traffic light. Start by turning on the current of the traffic light. The red light comes on first.



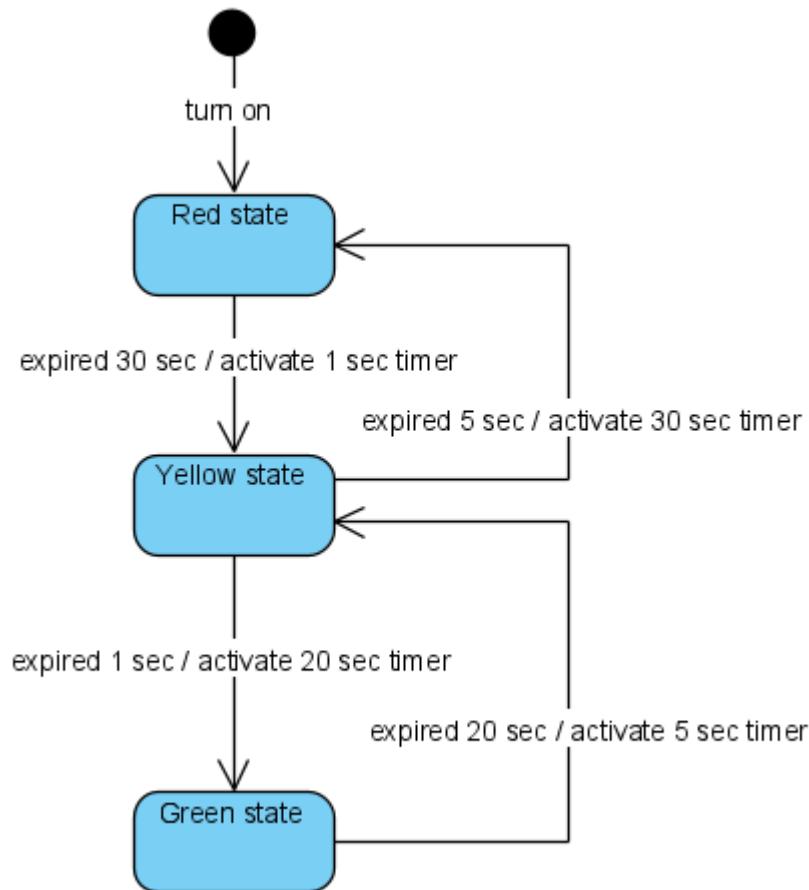
- The transition to yellow occurs after the 30 seconds have elapsed.



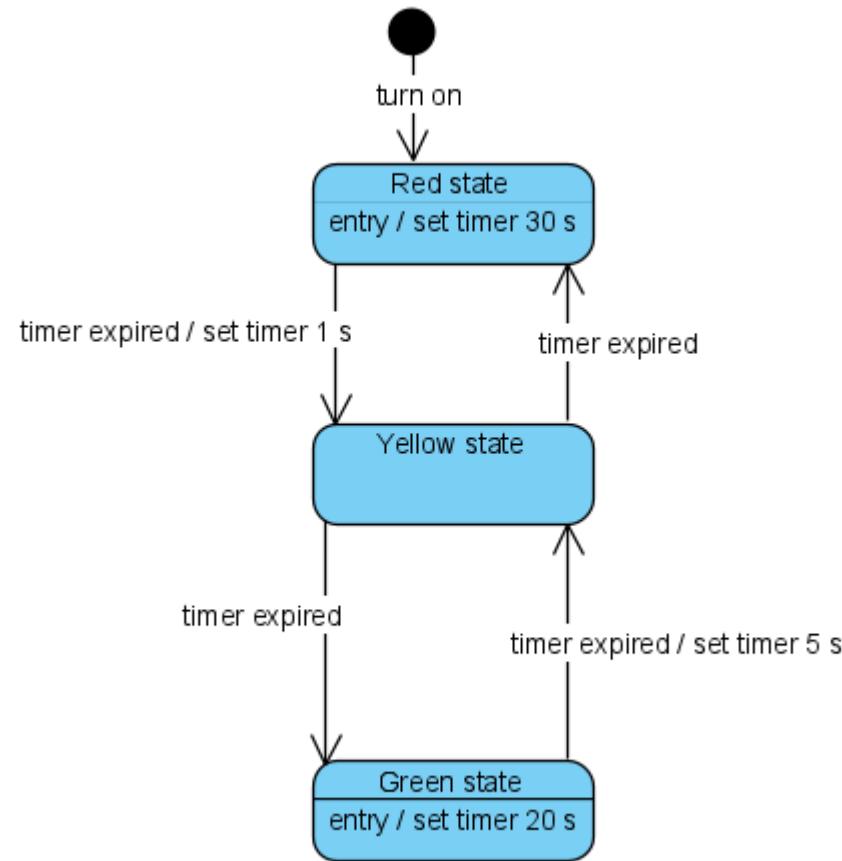
- After 1 second of the yellow signal, the green signal begins.



- The green signal goes out after 20 seconds and goes to yellow signal, which takes 5 seconds before lighting the red signal. After that the process repeats continuously.



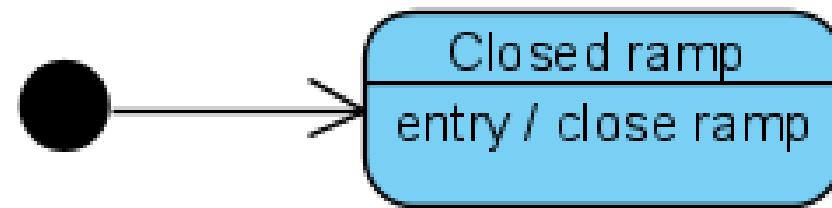
Solution 2



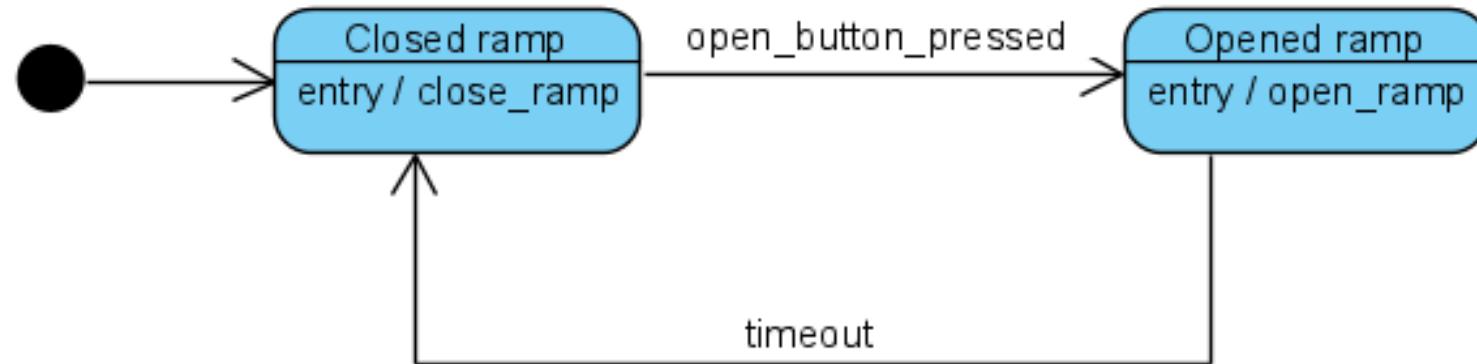
Example 2

- Model the passage of a car under a ramp in a parking lot using a state machine diagram. It all starts when the car comes up in front of a closed ramp. When the driver presses the Open button, the ramp switches to the open and remains opened 20 seconds. There is a sensor which monitors if the car is under the ramp. If the car gets stuck under the ramp, the ramp stays open, and if it goes through, the ramp goes to the final state.

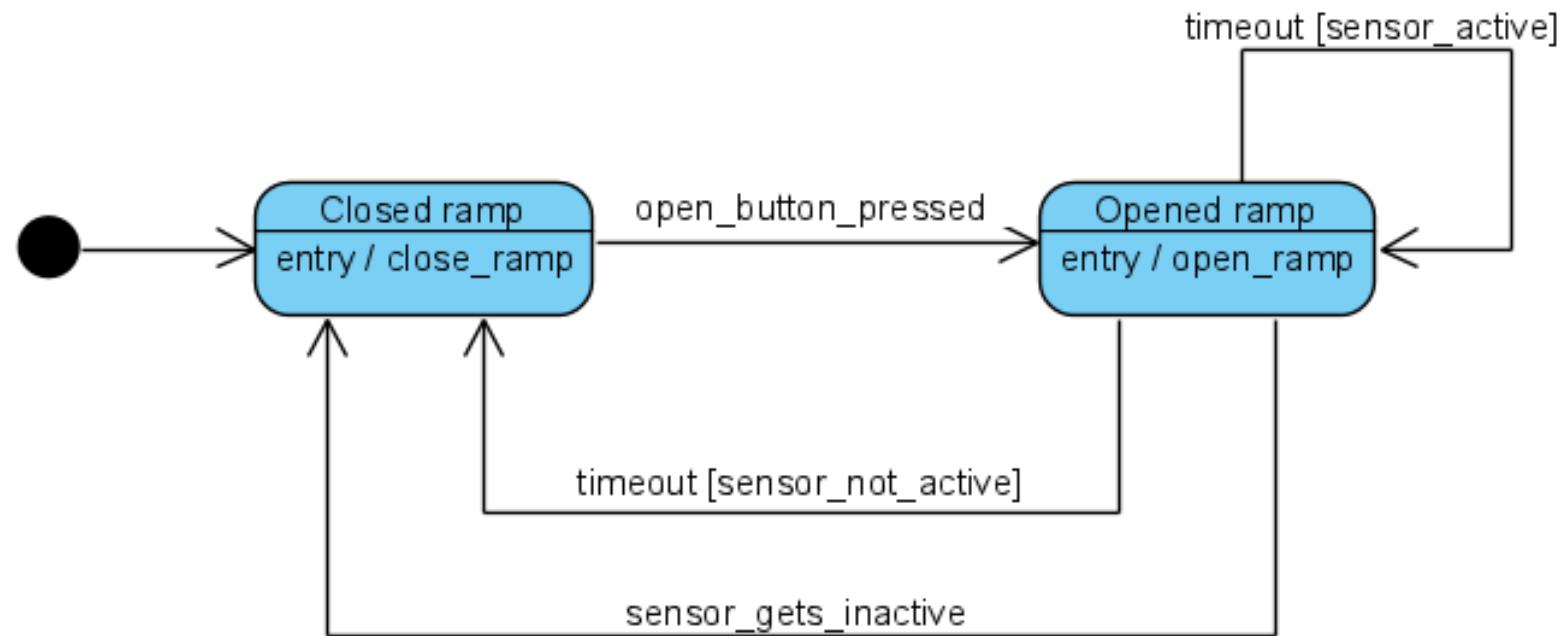
- Model the passage of a car under a ramp in a parking lot using a state machine diagram. It all starts when the car comes up in front of a closed ramp.



- When the driver presses the Open button, the ramp switches to the open and remains opened 20 seconds.



- There is a sensor which monitors if the car is under the ramp. If the car gets stuck under the ramp, the ramp stays open, and if it goes through, the ramp goes to the final state.



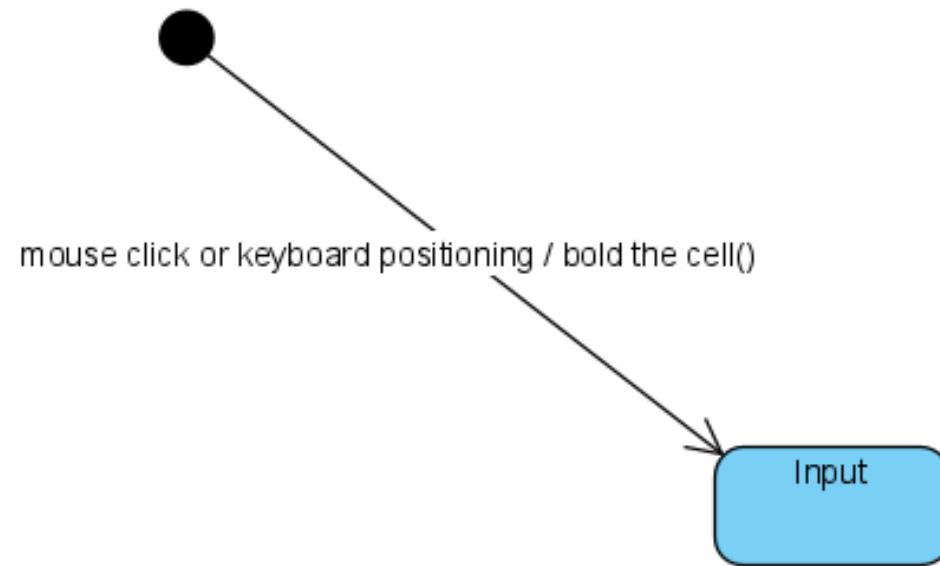
Example 3 [2]

Changing data in a cell

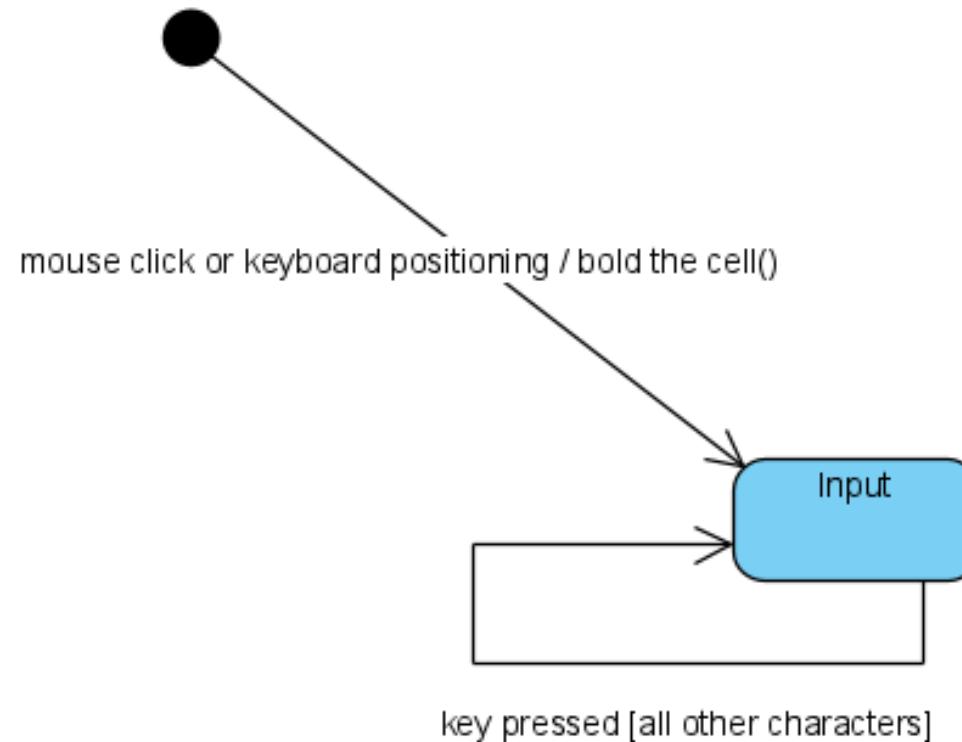
Model a state machine diagram to change data in a cell within a spreadsheet. Input begins with a mouse click or keyboard positioning, which causes the cell in the spreadsheet to be bolded on the screen and text can be entered. Assume that text entry lasts as long as any character other than Enter, Tab, or Esc is pressed. As you type, the character you press is printed on the screen, but the previous contents of the cell are not deleted.

When the input is successful ("Enter" or "Tab"), the data in the cell changes. First, the current contents of the cell are cleared from memory. Then, the current data is stored in memory and printed to a cell. If the entry is cancelled ("Esc"), the data in the cell is retained. In any case, the cells are defocused after the end of the entry, i.e. they are given a normal thickness.

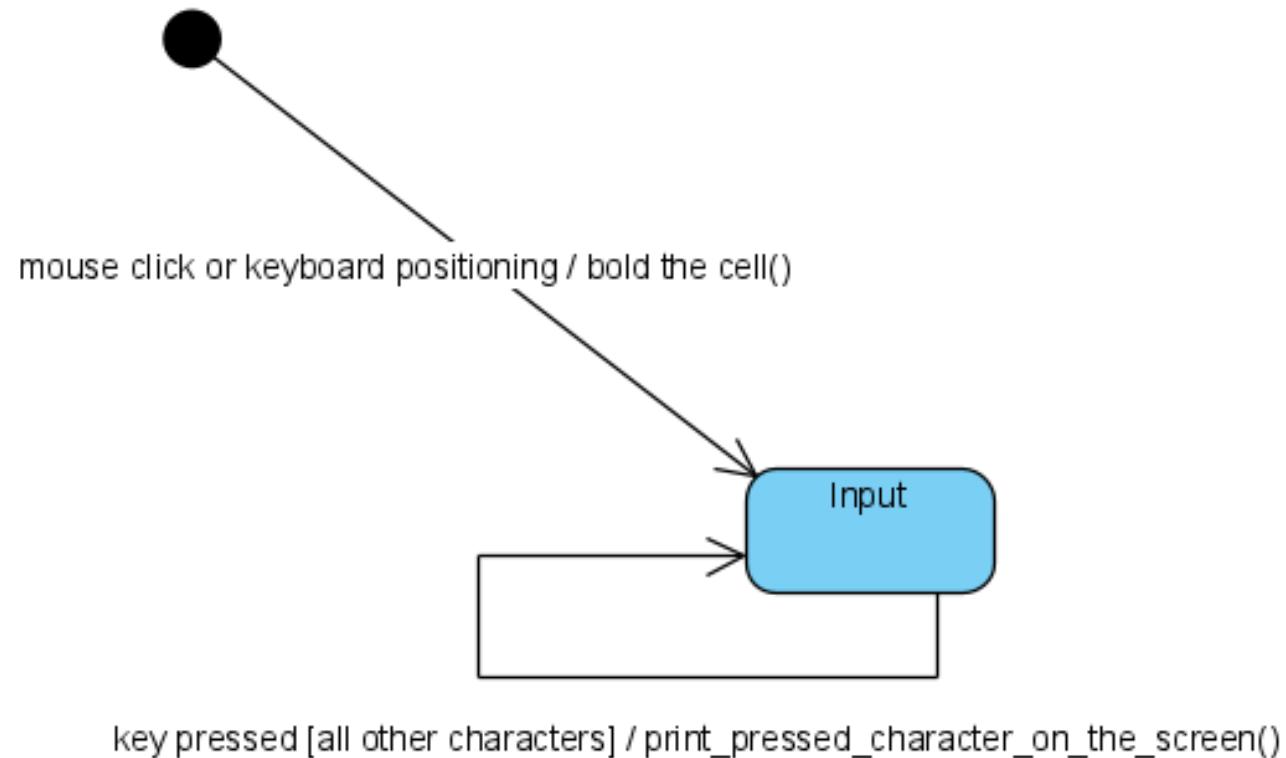
- Model a state machine diagram to change data in a cell within a spreadsheet. Input begins with a mouse click or keyboard positioning, which causes the cell in the spreadsheet to be bolded on the screen and text can be entered.



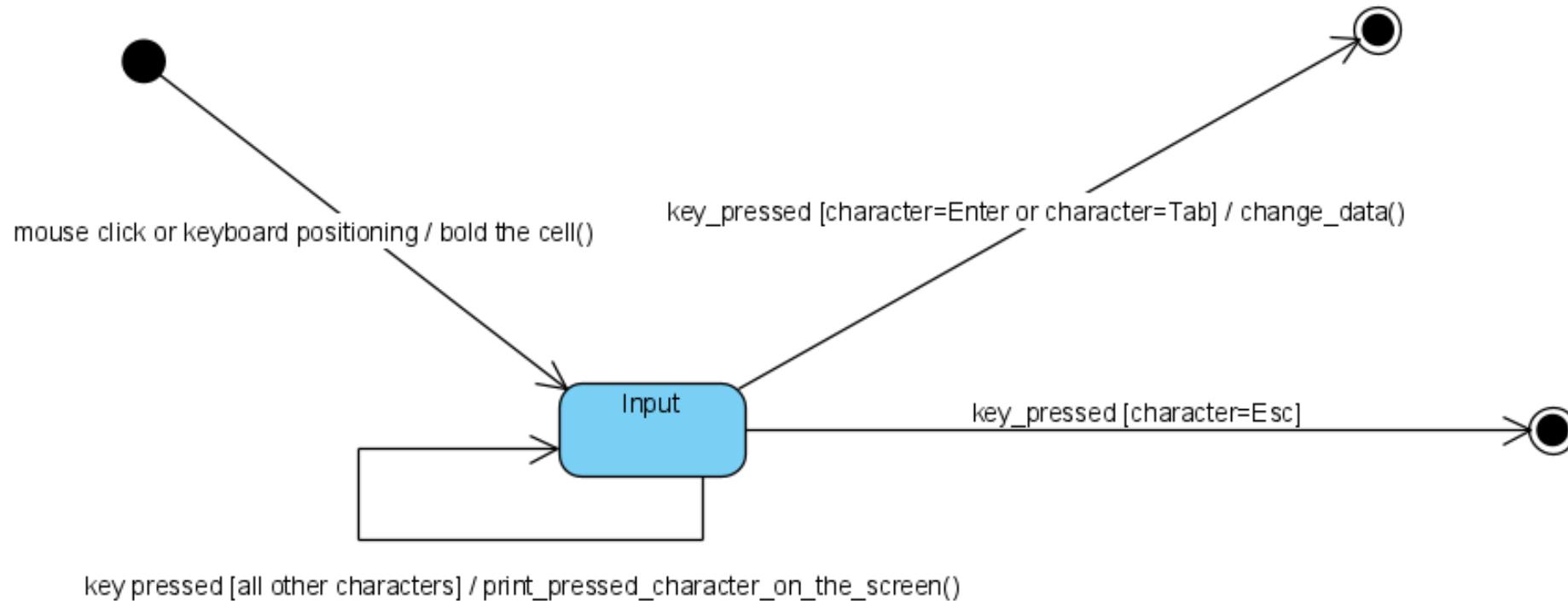
- Assume that text entry lasts as long as any character other than Enter, Tab, or Esc is pressed.



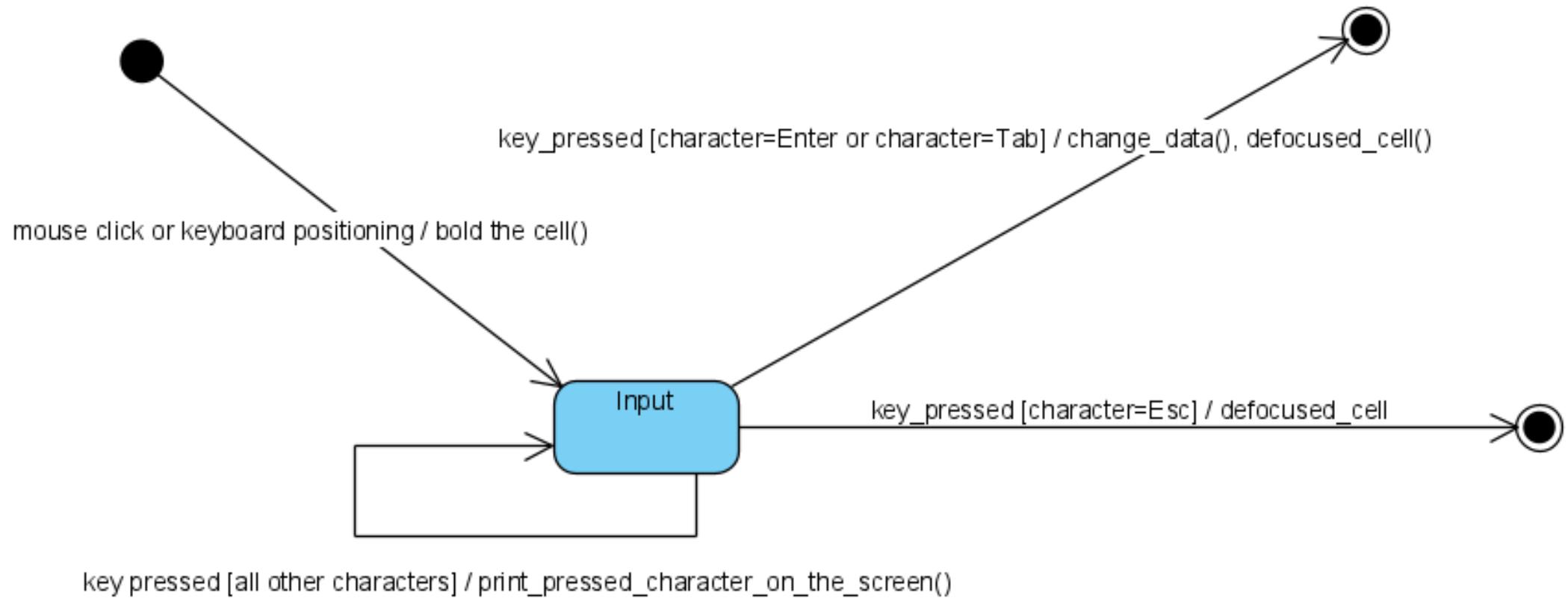
As you type, the character you press is printed on the screen, but the previous contents of the cell are not deleted.



- When the input is successful ("Enter" or "Tab"), the data in the cell changes. First, the current contents of the cell are cleared from memory. Then, the current data is stored in memory and printed to a cell. If the entry is cancelled ("Esc"), the data in the cell is retained.



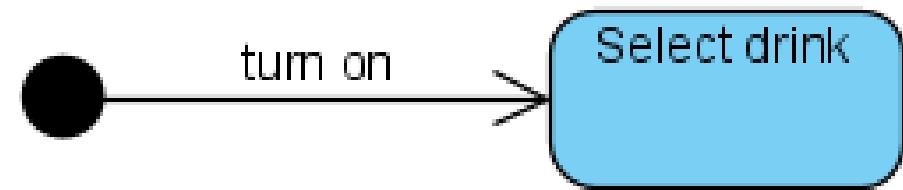
- In any case, the cells are defocused after the end of the entry, i.e. they are given a normal thickness.



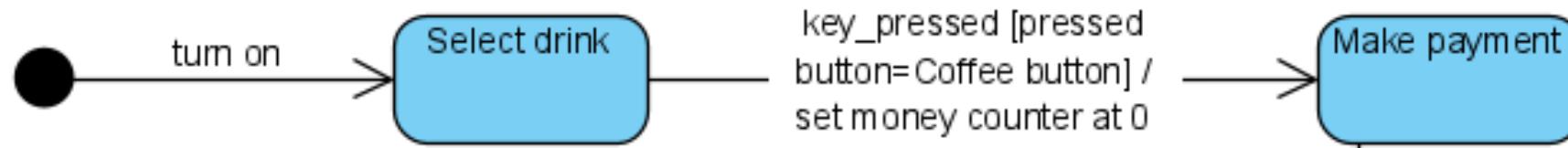
Example 4

- **Model the state machine diagram of the coffee maker.** Begin by turning on the coffee maker. When the coffee maker is on, you can see all the options on the screen. When you press the coffee button, the payment process will start and money counter will be at 0. You have to insert coins. The amount of money inserted is displayed on the screen until you insert the same amount or more than coffee price. Then the coffee machine will start making your coffee. This will take 30 seconds. After that you will see on the screen the messages: "Take out cup of coffee" and "Take out excess money" and at the same time the coffee machine will return the excess money.

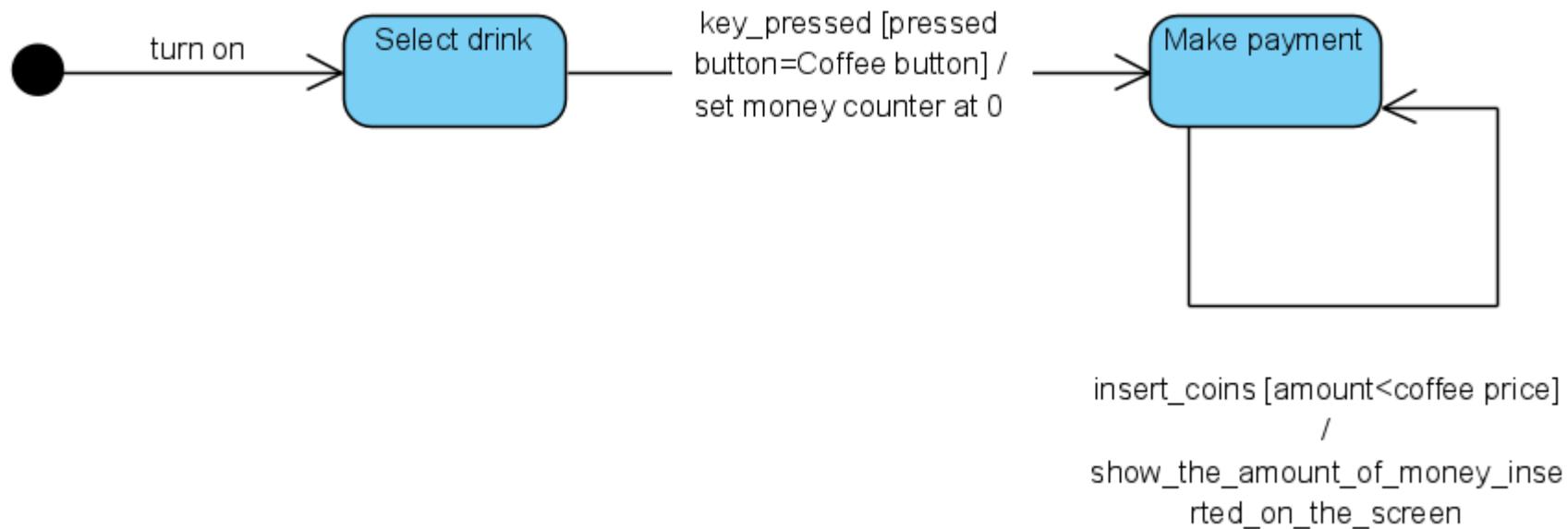
- Begin by turning on the coffee maker. When the coffee maker is on, you can see all the options on the screen.



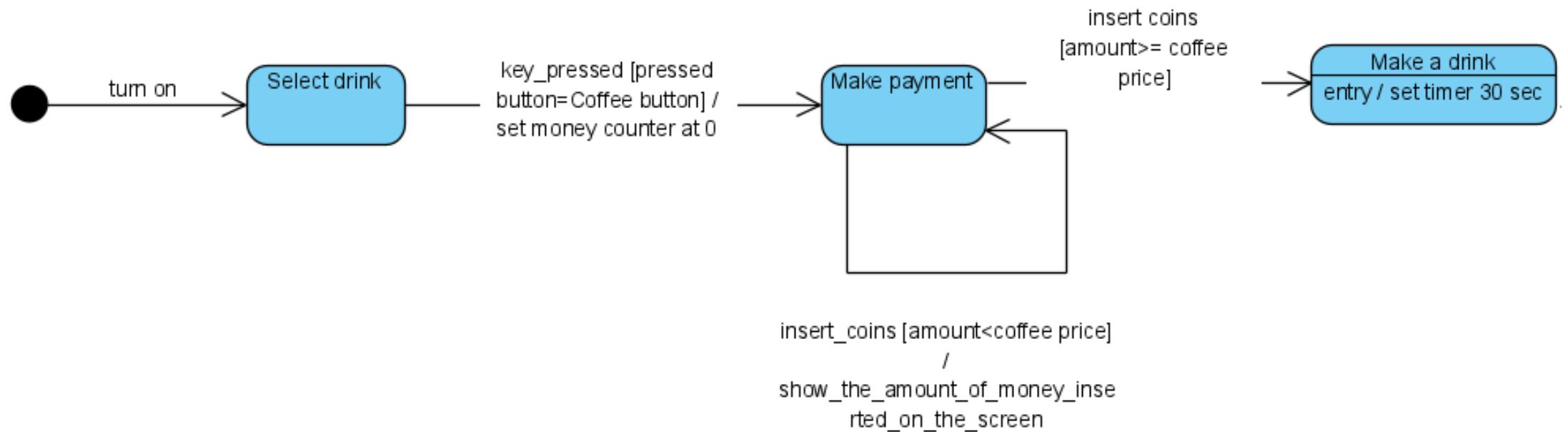
- When you press the coffee button, the payment process will start and money counter will be at 0.



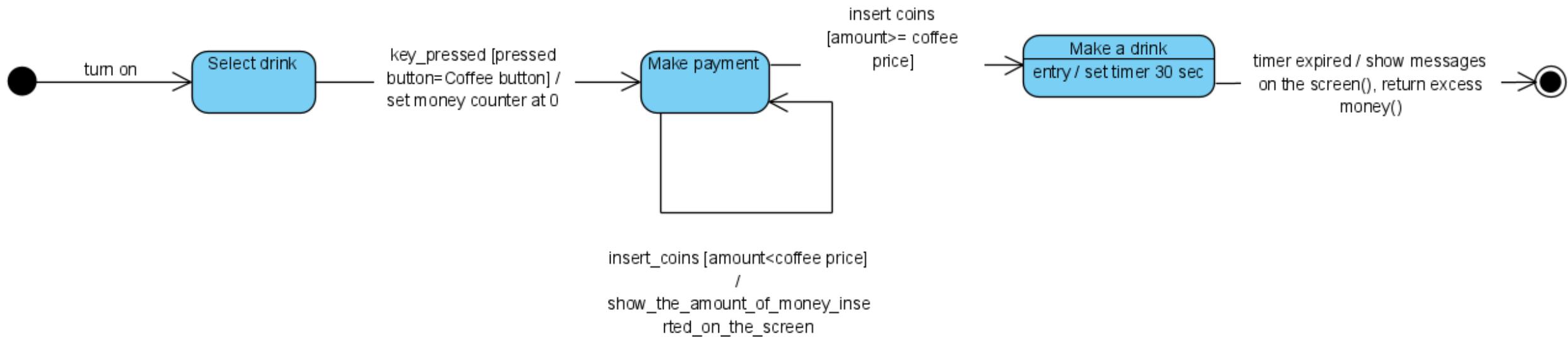
- You have to insert coins. The amount of money inserted is displayed on the screen until you insert the same amount or more than coffee price.



- Then the coffee machine will start making your coffee. This will take 30 seconds.



- After that you will see on the screen the messages: "Take out cup of coffee" and "Take out excess money" and at the same time the coffee machine will return the excess money.



Homework assignment

- Update the use case descriptions individually. Describe each use case description using one of the UML diagrams (Activity diagram, State machine diagram, Sequence diagram,...).
- As a group, you need to upload a pdf file with all the use case descriptions and the UML diagrams that describes them (emphasise which author wrote which use case description and UML diagram).

Literature

- [1] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/>
- [2] Jović, A., Horvat, M. & Grudenić, I. (2014) UML-dijagrami: Zbirka primjera i riješenih zadataka. Zagreb. GRAPHIS d.o.o. Zagreb.

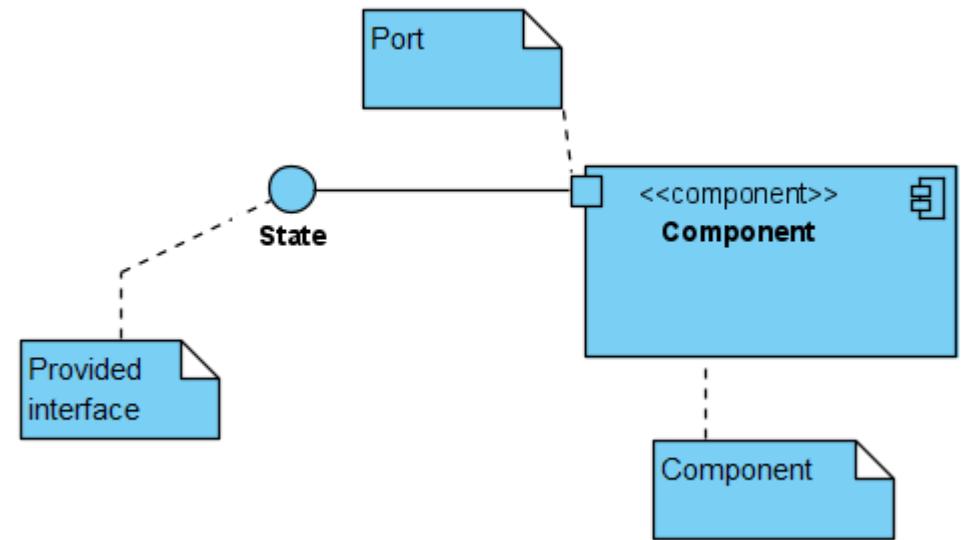
UML component diagram

Definition

- **Component diagram** breaks down the system into smaller components to depict the system architecture.
- Component diagram shows components and their connections.
- A component is a modular unit with clear interfaces and corresponding ports.
- Some components offer ports while others may need them for their operation.
- Internal properties of components are hidden and inaccessible.

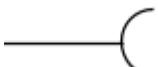
Component Diagram Notation

- Provided interfaces define "a set of public attributes and operations that must be provided by the classes that implement a given interface". [2]
- A port (definition) indicates that the component itself does not provide the required interfaces (e.g., required or provided). Instead, the component delegates the interface(s) to an internal class. [2]



Relationships

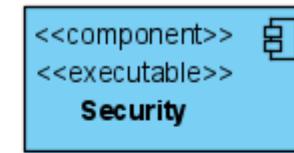
- Dependency [3]
 - A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
 - This means that the complete semantics of the depending elements is either semantically or structurally dependent on the definition of the supplier element(s).



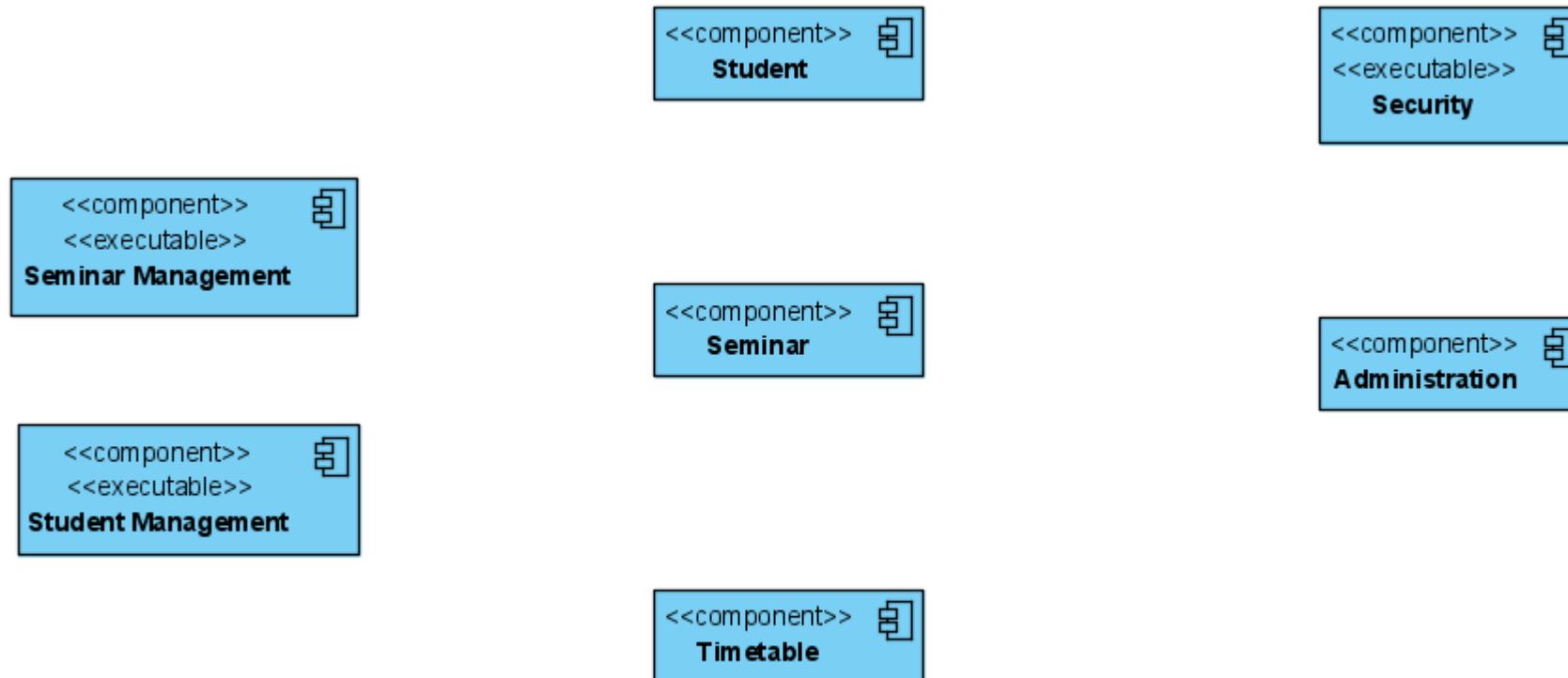
Example 1. [1]

- The multi-tier university information system consists of three major subsystems: Seminar Management, Student Management, and Security, which correspond to executable files in the model. In the top layer of the system are the ManagementSeminars and ManagementStudents components, in the second layer Student, Seminar and Timetable, and in the third layer Security and Administration. In the second layer, the Student component has the IDB and IStudent interfaces, the Seminar component has the IDB and ISeminar interfaces, and the Timetable component has the IDB and ISchedule interface. In the third layer, the Security component has the interfaces IPassword and IAccess, and the Administration component has the interface IAdministration. The Seminar Management component imports the ISchedule and ISeminar interfaces, while the Student Management component imports the same interfaces plus IStudent. The Student, Seminar and Timetable components use the IAccess and IAdministration interfaces. Administration component access the database using the ADO.NET framework. Display the corresponding component diagram.

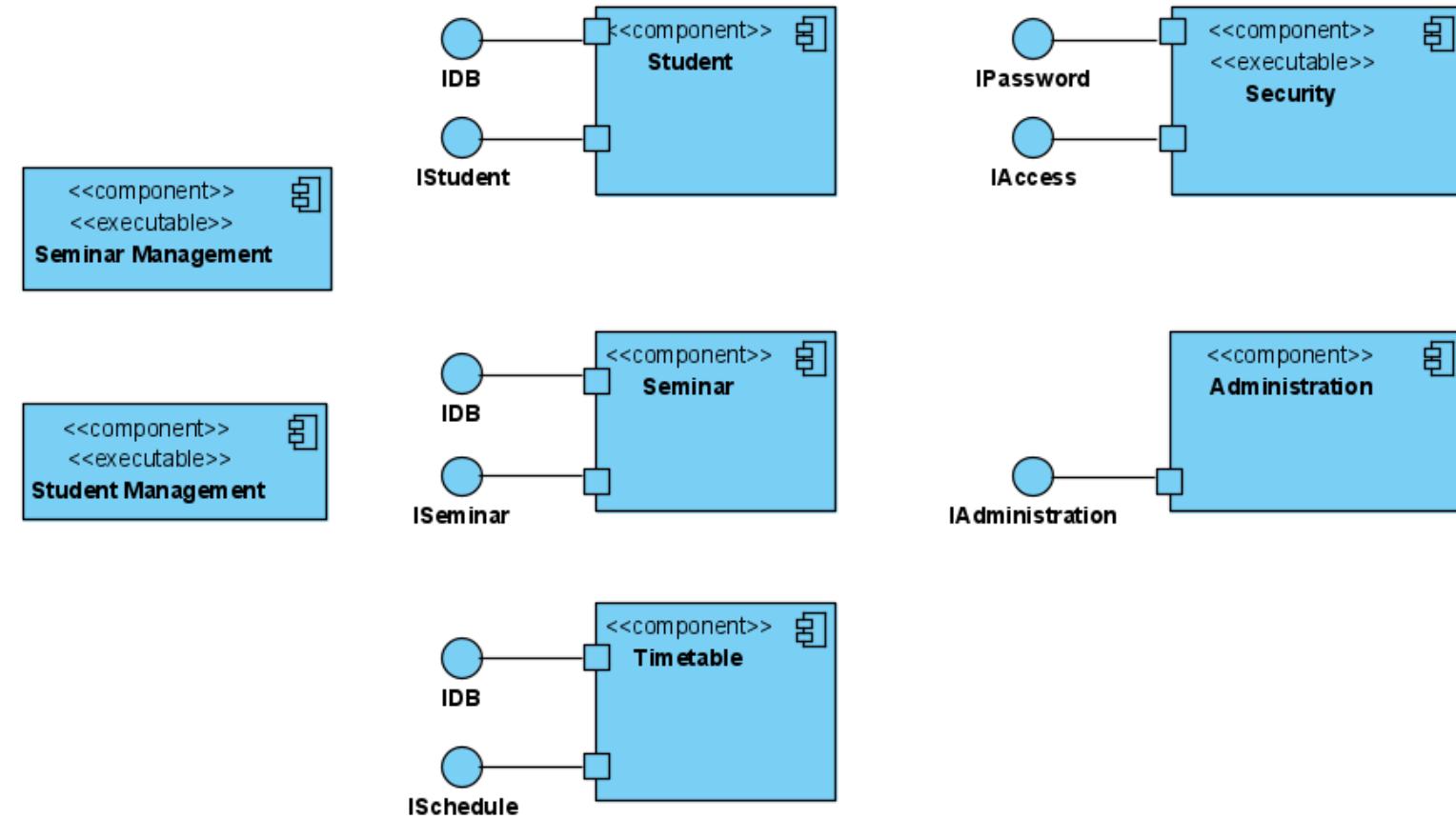
- The multi-tier university information system consists of three major subsystems: Seminar Management, Student Management, and Security, which correspond to executable files in the model.



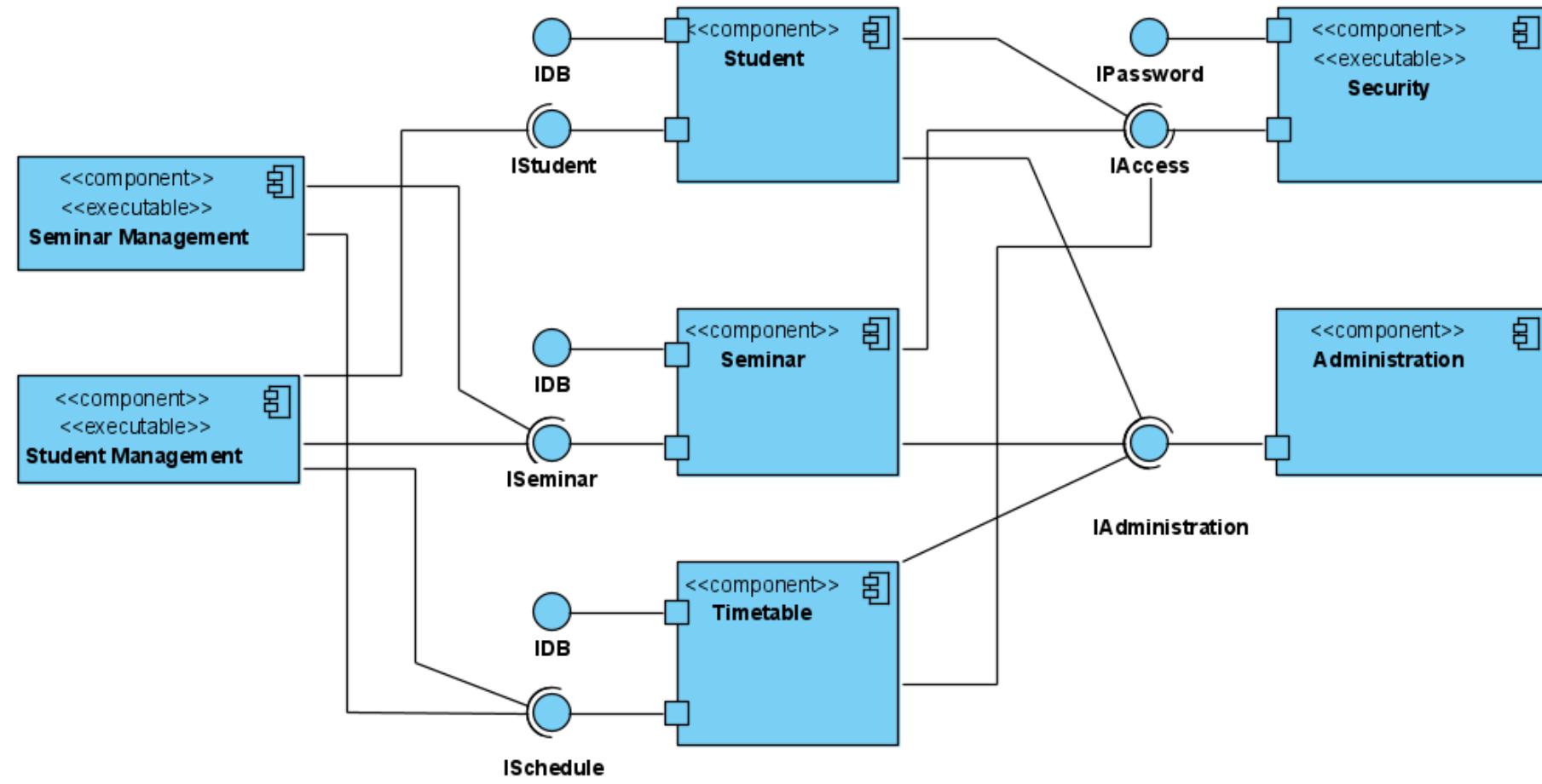
- In the top layer of the system are the ManagementSeminars and ManagementStudents components, in the second layer Student, Seminar and Timetable, and in the third layer Security and Administration.



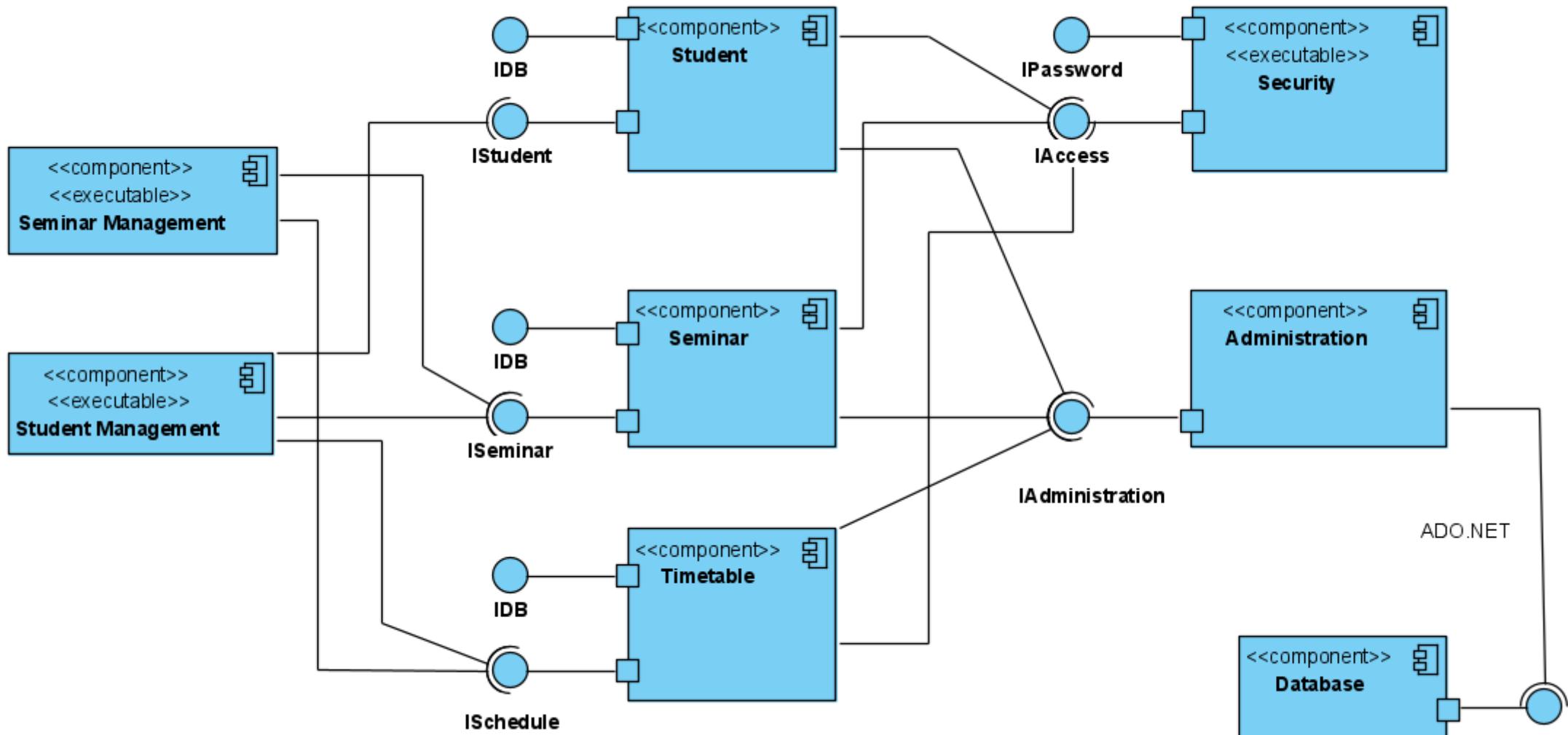
- In the second layer, the Student component has the IDB and IStudent interfaces, the Seminar component has the IDB and ISeminar interfaces, and the Timetable component has the IDB and ISchedule interface. In the third layer, the Security component has the interfaces IPassword and IAccess, and the Administration component has the interface IAdministration.



The Seminar Management component imports the ISchedule and ISeminar interfaces, while the Student Management component imports the same interfaces plus IStudent. The Student, Seminar and Timetable components use the IAccess and IAdministration interfaces.



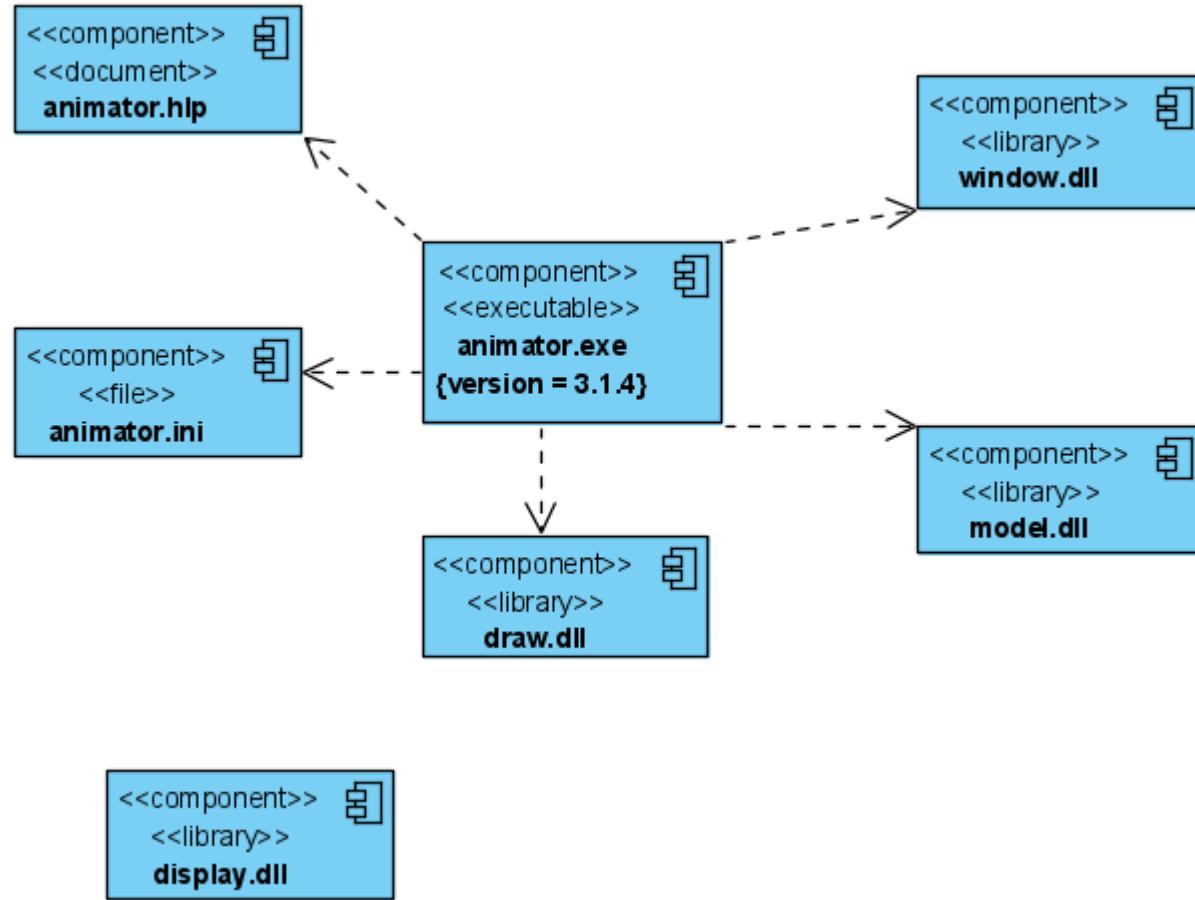
Administration component access the database using the ADO.NET framework.



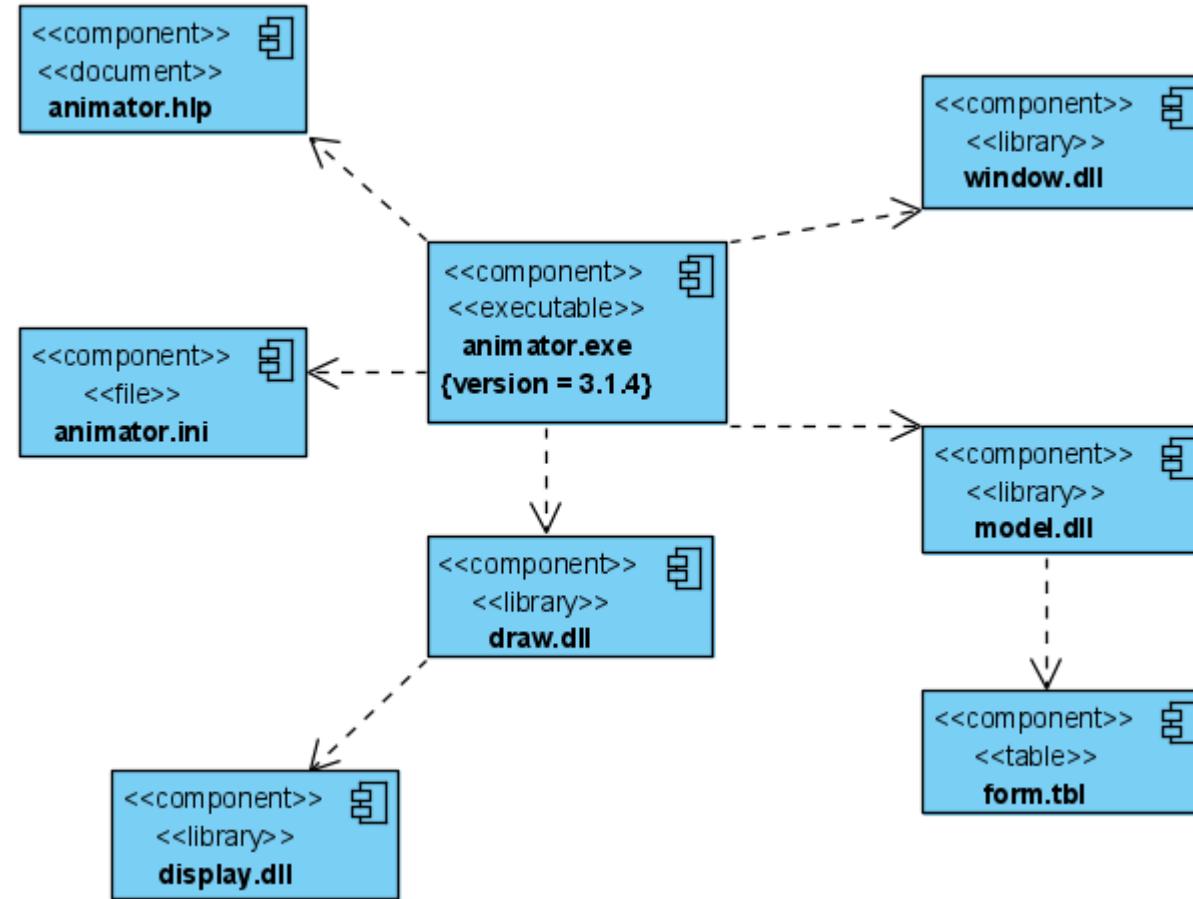
Example 2. [1]

- After installation, the Windows animation application "Animator" consists of an executable file "animator.exe", which uses the DLL files "window.dll", "model.dll", "draw.dll" and "display.dll". The application configuration is in the formatted file "animator.ini" and the help system is in the binary file "animator.hlp." The executable file "animator.exe" is otherwise 3.1.4. The library "model.dll" stores data in a database table called "form", which consists of the file "form.tbl". The library "drawing.dll" implements the "display.dll". Show the listed components in the diagram.

After installation, the Windows animation application "Animator" consists of an executable file "animator.exe", which uses the DLL files "window.dll", "model.dll", "draw.dll" and "display.dll". The application configuration is in the formatted file "animator.ini" and the help system is in the binary file "animator.hlp." The executable file "animator.exe" is otherwise 3.1.4.

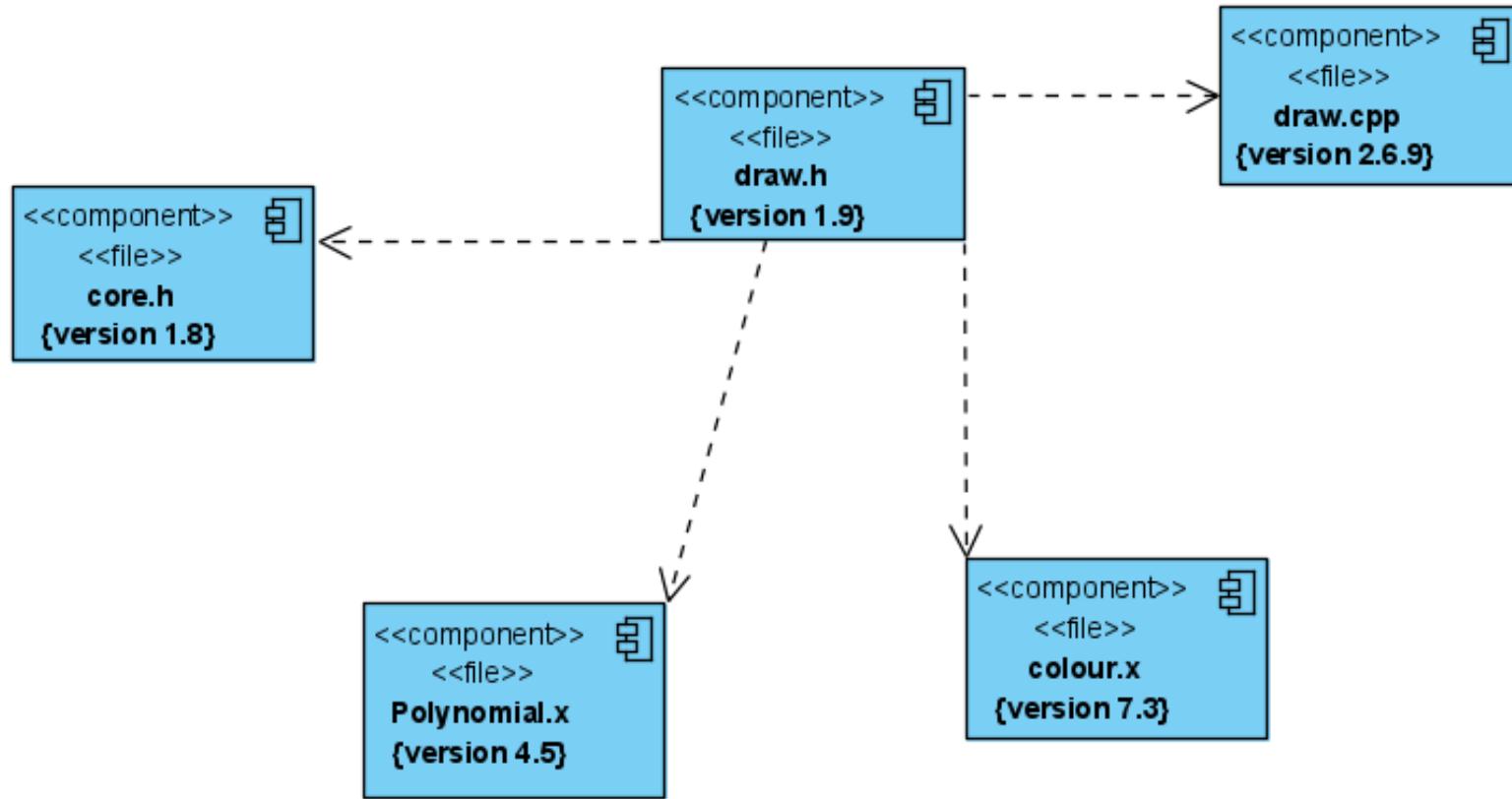


- The library "model.dll" stores data in a database table called "form", which consists of the file "form.tbl". The library "drawing.dll" implements the "display.dll".



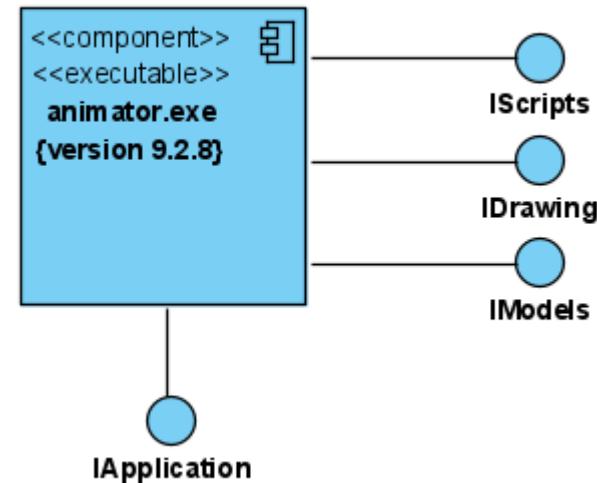
Example 3. [1]

- A computer graphics project in the C ++ programming language consists of several source code files and headers: "draw.h" (version 1.9), "draw.cpp" (version 2.6.9), "core.h" (version 1.8) , "Polynomial.x" (version 4.5) and "colour.x" (version 7.3). The "draw.h" **file depends on all other files**. Display the corresponding component diagram.



Example 4. [1]

- The animation application has its own API - the framework defines four interfaces: IScripts, IDrawing, IMODELS and IApplication. The application executable is "animator.exe" in version 9.2.8. Display a component diagram.



UML deployment diagram

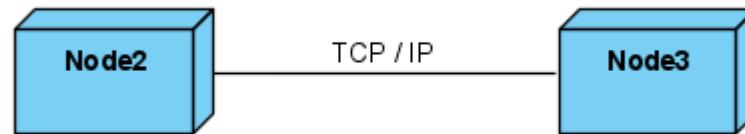
Definition

- Deployment diagram is used to visualize the relation between software and hardware.
- Deployment diagram shows physical relations between system HW and SW.
- HW elements:
 - Computers (servers, clients)
 - Embedded devices
 - Other devices (sensors, periphery)
- Deployment diagram shows HW used by the SW.

Deployment Diagram Notation

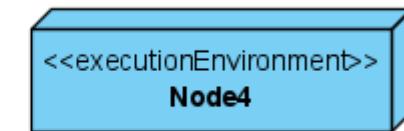
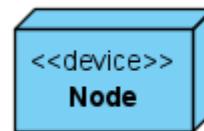
Deployment diagram elements

- Nodes [4]
 - A node usually represents a piece of hardware in the system. A connection depicts the communication path used by the hardware to communicate usually indicates the method i.e. TCP/IP



Nodes [5]

- There are two types of nodes in a deployment diagram: device nodes and execution environment nodes. Device nodes are computing resources with processing capabilities and the ability to execute programs. Some examples of device nodes include PCs, laptops, and mobile phones.
- An execution environment node, or EEN, is any computer system that resides within a device node. It could be an operating system, a JVM, or another servlet container.



- **Artifact:** A product developed by the software, symbolized by a rectangle with the name and the word “artifact” enclosed by double arrows. [5]
- An artifact is the specification of a physical piece of information, such as, source files, binary executable files, table in a database system. [4]
- An artifact is defined by the user represents a concrete element in the physical world. [4]

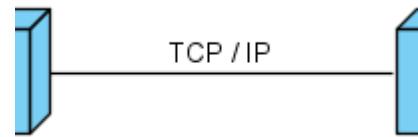


- **Component:** A rectangle with two tabs that indicates a software element. [5]
- **Component** - An entity required to execute a stereotype function. [5]



Association

- **Association:** A line that indicates a message or other type of communication between nodes. [5]
- A straight line that represents communication between two device nodes. [5]



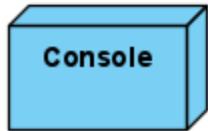
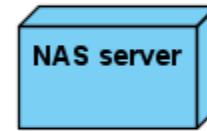
- **Dependency:** A dashed line that ends in an arrow, which indicates that one node or component is dependent on another. [5]



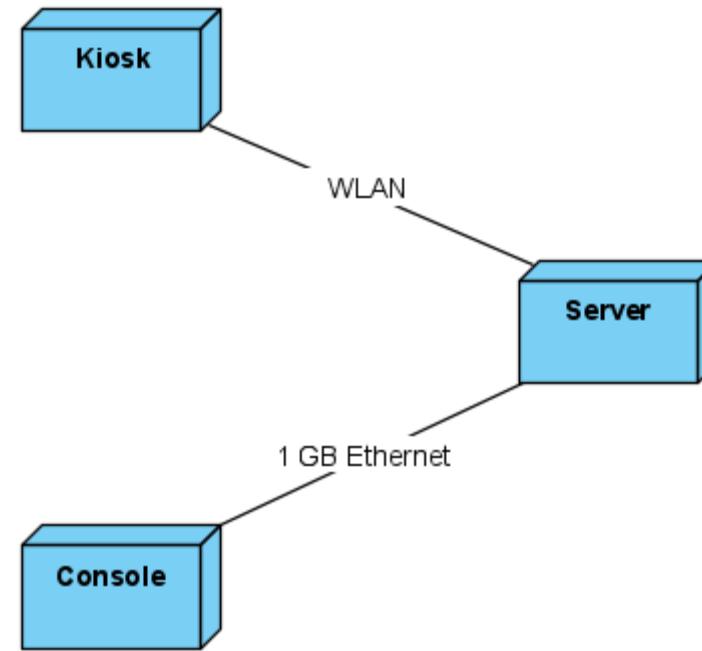
Example 4 [1] Links between – hardware components

- A system consists of multiple nodes: a kiosk, a console, a server, a NAS (Network-attached storage) storage server, and a hard disk RAID (Redundant Array of Independent Disks) array. The kiosk is wirelessly connected to the server via WLAN and the console is connected via 1 GB Ethernet. The server stores the data on the hard disk array via a NAS server. Model the deployment diagram.

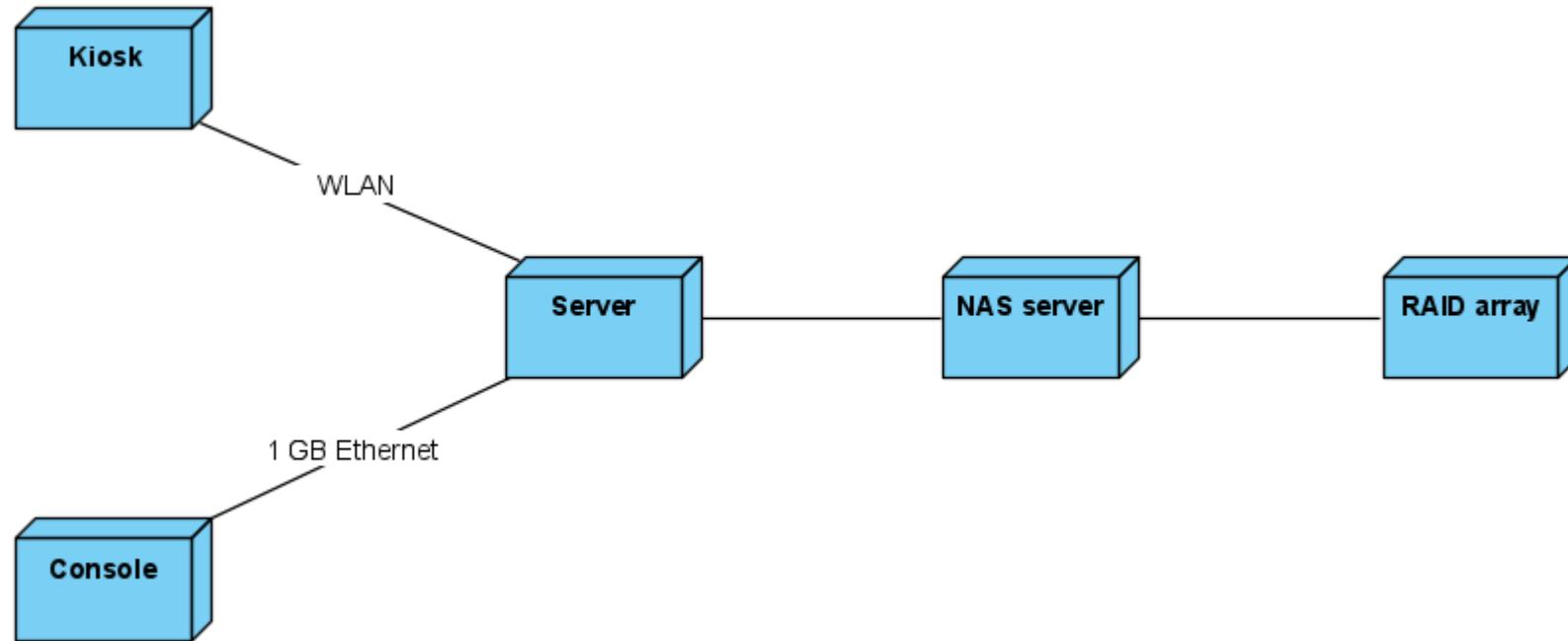
- A system consists of multiple nodes: a kiosk, a console, a server, a NAS (Network-attached storage) storage server, and a hard disk RAID (Redundant Array of Independent Disks) array.



- The kiosk is wirelessly connected to the server via WLAN and the console is connected via 1 GB Ethernet.



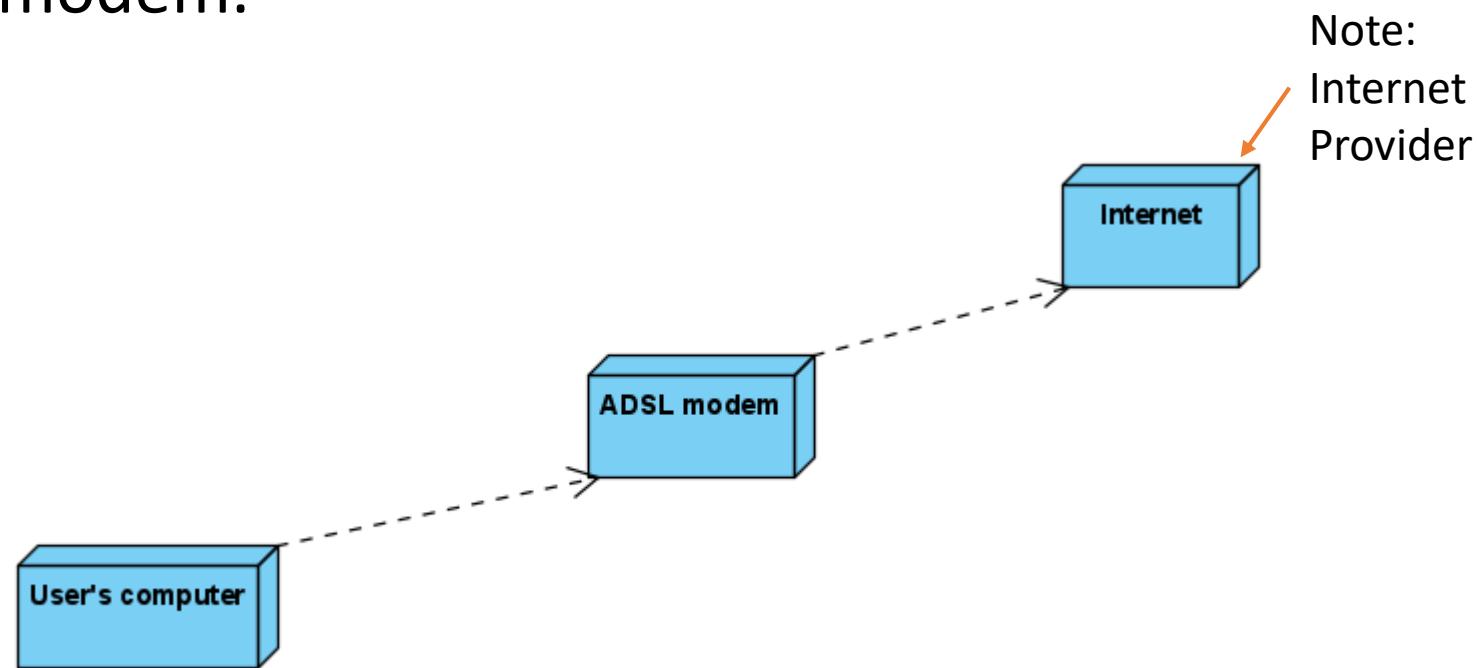
- The server stores the data on the hard disk array via a NAS server.



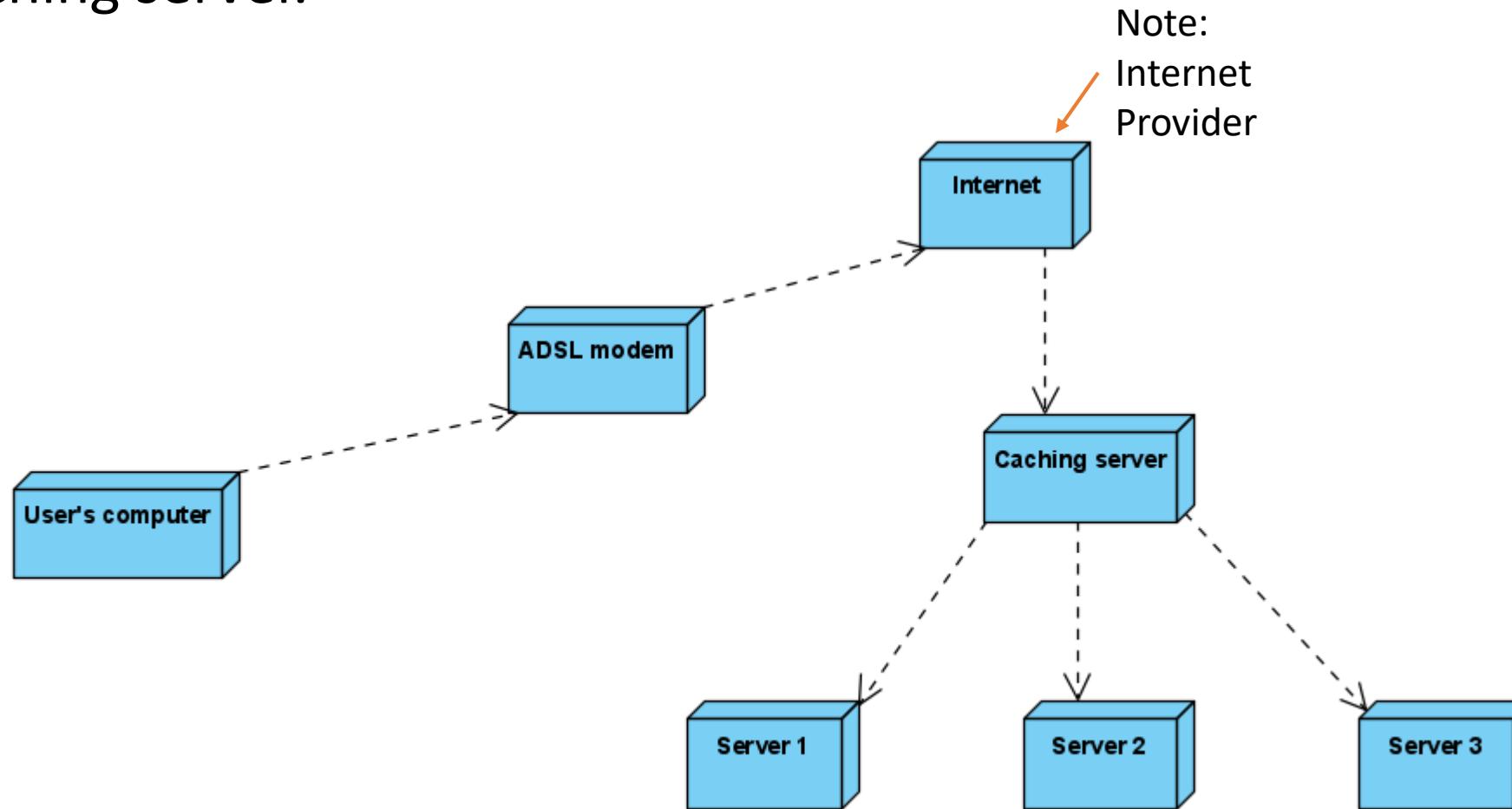
Example 5 [1] Dependency between hardware components

- A two-tier information system consists of a user application that connects to a server application over the Internet Provider. The user application runs on the user's computer and connects to the Internet via an ADSL modem. The server application runs on a cluster of three servers managed by a caching server. Model the deployment diagram.

- A two-tier information system consists of a user application that connects to a server application over the Internet Provider. The user application runs on the user's computer and connects to the Internet via an ADSL modem.

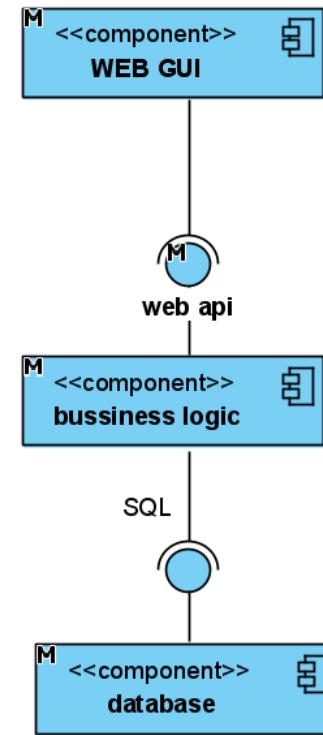


- The server application runs on a cluster of three servers managed by a caching server.

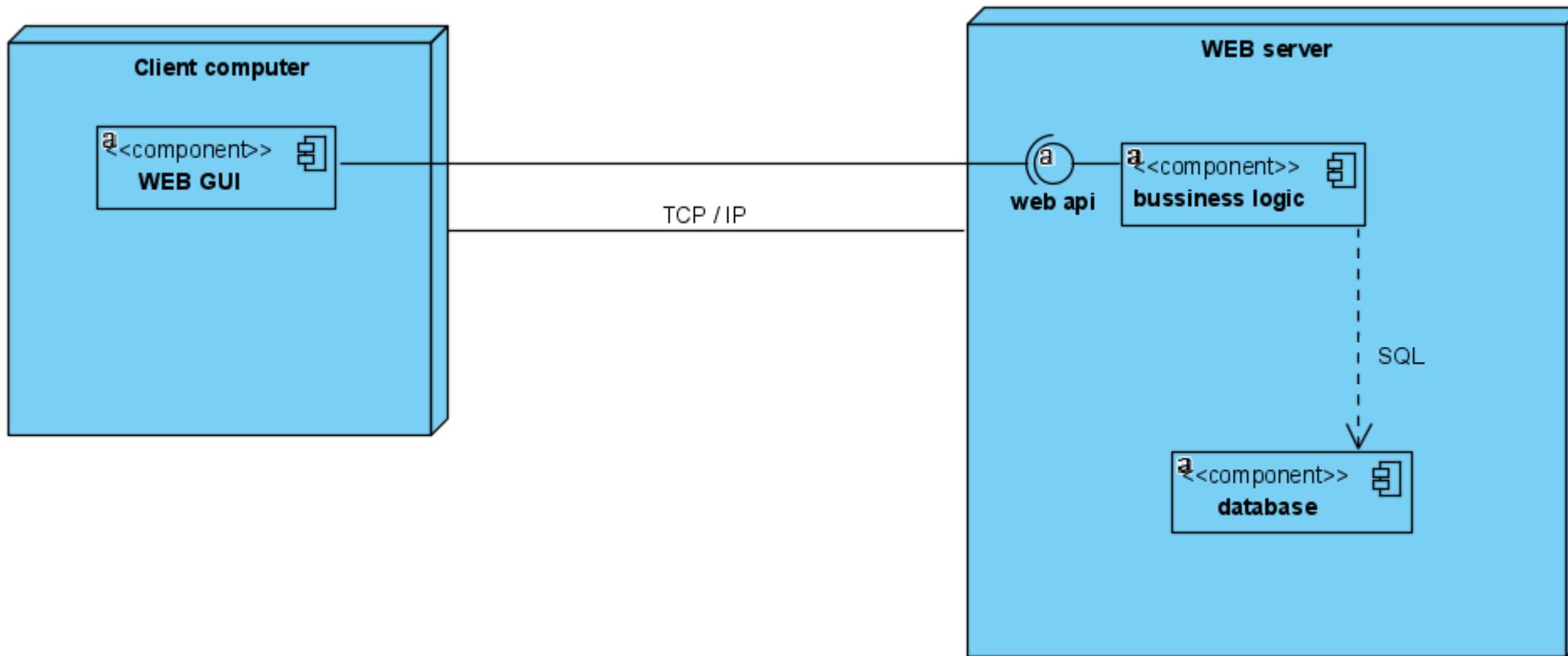


Example 6: a common practical example

- Component diagram



Deployment diagram



Homework assignment

- You must create a Software Requirements Specifications (SRS) for your proposed project.
 - **Software Requirements Specifications (SRS):**
 - 1. Presentation of the problem
 - 2. Brief presentation of the solution
 - 3. Use cases
 - 3a – Use Case diagram
 - 3b – Use case descriptions - requirements- Functional and non-functional requirements. (https://en.wikipedia.org/wiki/Non-functional_requirement)
 - 3c - Additional UML diagrams explaining UC
 - 4. GUI - sketch of user interface (<https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/>)
 - 5. Connections to other systems – API
 - 6. miscellaneous
-
- You all need to submit the SRS on time because next time you will get "review reports" from another group for the homework assignment.
 - **If you do not submit on time, your final grade will be 0 for the next two homework assignments.**
 - **Both assignments will be graded at once, but you must submit each assignment by the deadline or the grade will be 0.**

Literature

- [1] Jović, A., Horvat, M. & Grudenić, I. (2014) UML-dijagrami: Zbirka primjera i riješenih zadataka. Zagreb. GRAPHIS d.o.o. Zagreb.
- [2] <https://online.visual-paradigm.com/diagrams/tutorials/component-diagram-tutorial/>
- [3] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/>
- [4] <https://www.visual-paradigm.com/learning/handbooks/software-design-handbook/deployment-diagram.jsp>
- [5] https://www.lucidchart.com/pages/uml-deployment-diagram/#section_4
- [6] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>

UML class diagram

UML class diagram - definition

- **Class diagram** - depicts classes, alongside with their attributes and their behaviors .

UML class diagram composition

- The UML class diagram is composed of:
 - A set of classes
 - A set of relationships between classes

Class Notation

- A class notation consists of three parts:
 - 1. Class Name
 - 2. Class attributes
 - 3. Class operations
- Visibility of Class attributes and Operations
 - + denotes public attributes or operations
 - - denotes private attributes or operations
 - # denotes protected attributes or operations

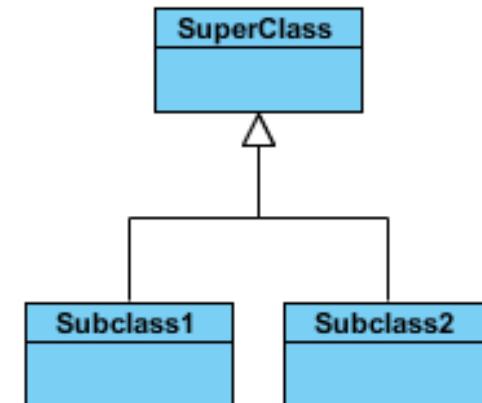
Class Name
+attribute1 : int
-attribute2 : float
#attribute3 : Circle
+operation1() : string
-operation2() : float

Class relationships

- Relationship Type
 - Inheritance (or Generalization)
 - Simple Association
 - Aggregation
 - Composition
 - Dependency

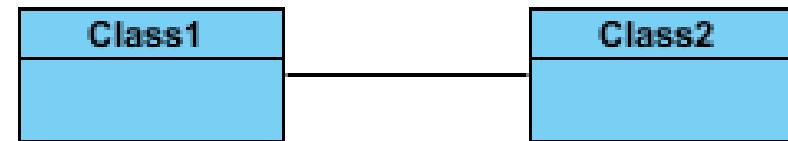
Inheritance (or Generalization): [3]

- Represents an "is-a" relationship.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class



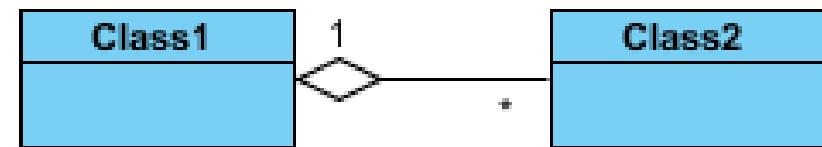
Simple Association: [3]

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



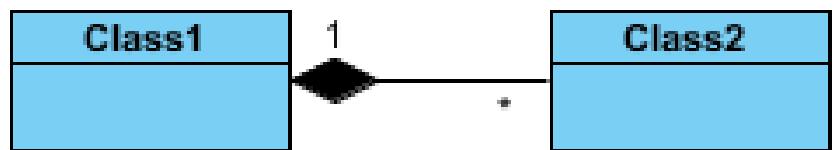
Aggregation: [3]

- A special type of association. It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite



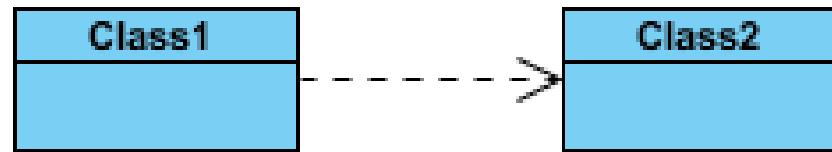
Composition: [3]

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite



Dependency: [3]

- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow



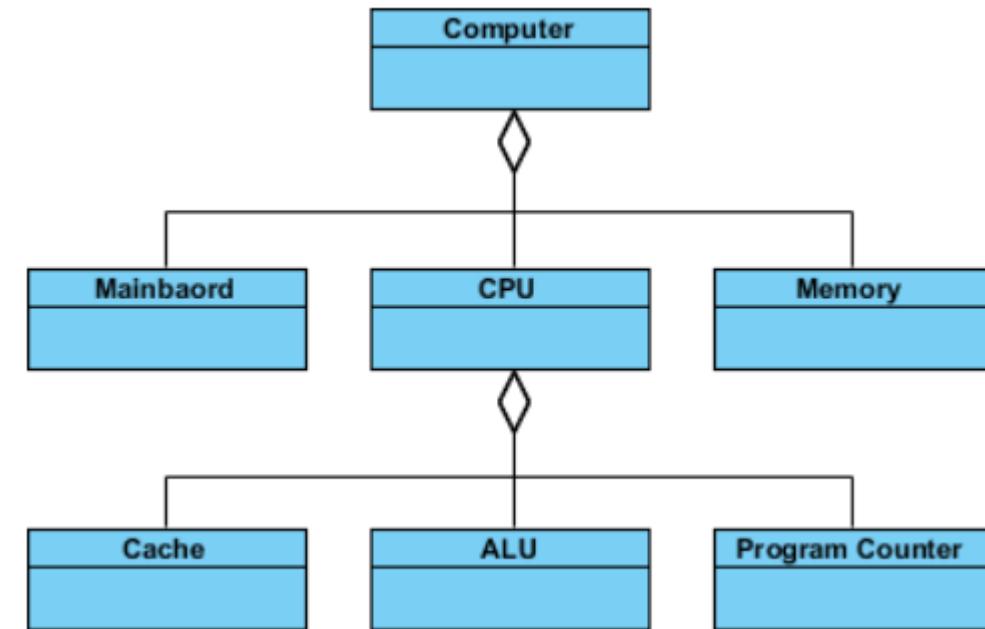
Multiplicity [3, 4]

- How many objects of each class take part in the relationships and multiplicity can be expressed as:
 - Exactly one - 1
 - Zero or one - 0..1
 - Many - 0..* or *
 - One or more - 1..*
 - Exact Number - e.g. 3..4 or 6
 - Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5



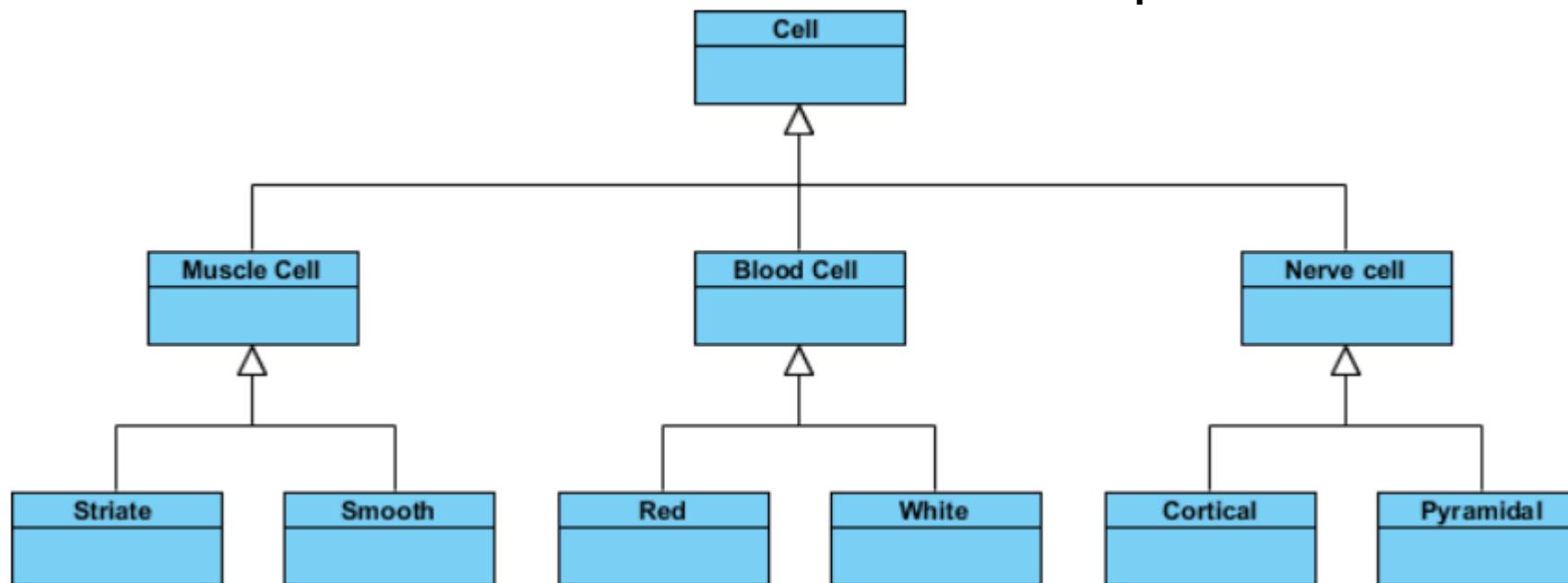
Aggregation Example - Computer and parts [3]

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes
- ALU (Arithmetic logic unit)



Inheritance Example - Cell Taxonomy [3]

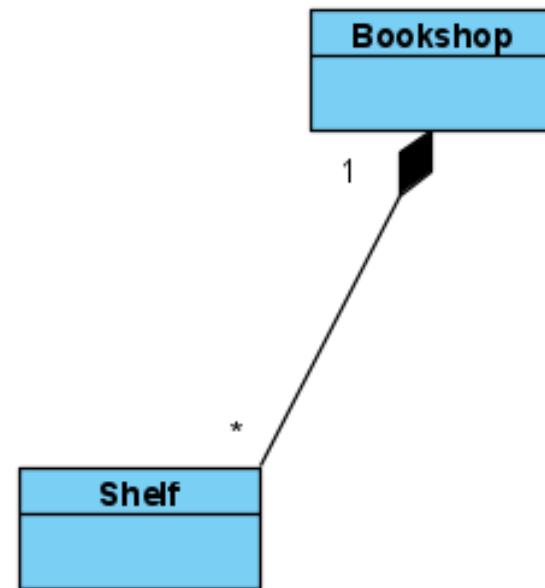
- Inheritance is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The child classes inherit the attributes and operations of the parent class.



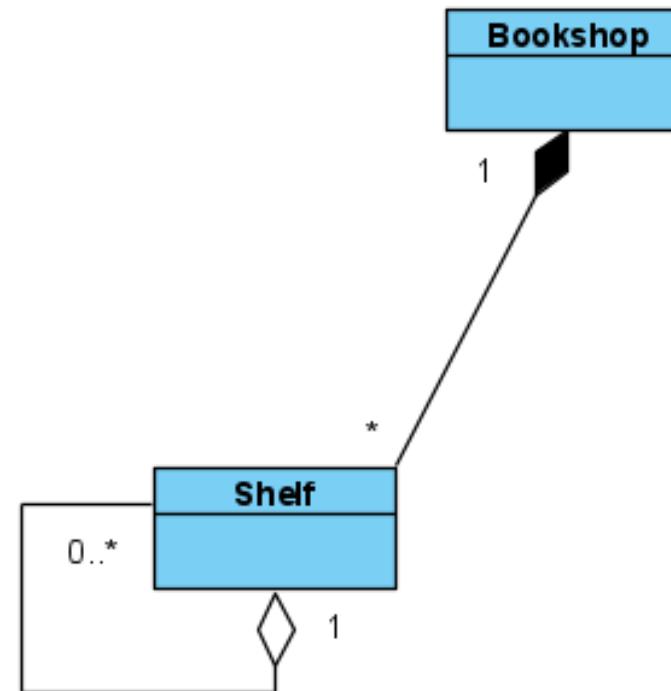
Example 1 [1]

- In a bookshop there are several shelves which are arranged thematically, that is, within the shelves there are shelves, within the shelves there are other shelves, and so on. On each shelf there are some books. When the bookstore is restructured, the books are distributed to other shelves. Each book has its own ISBN (string), title (string), publisher (string) and publication date (date). Make a class diagram.

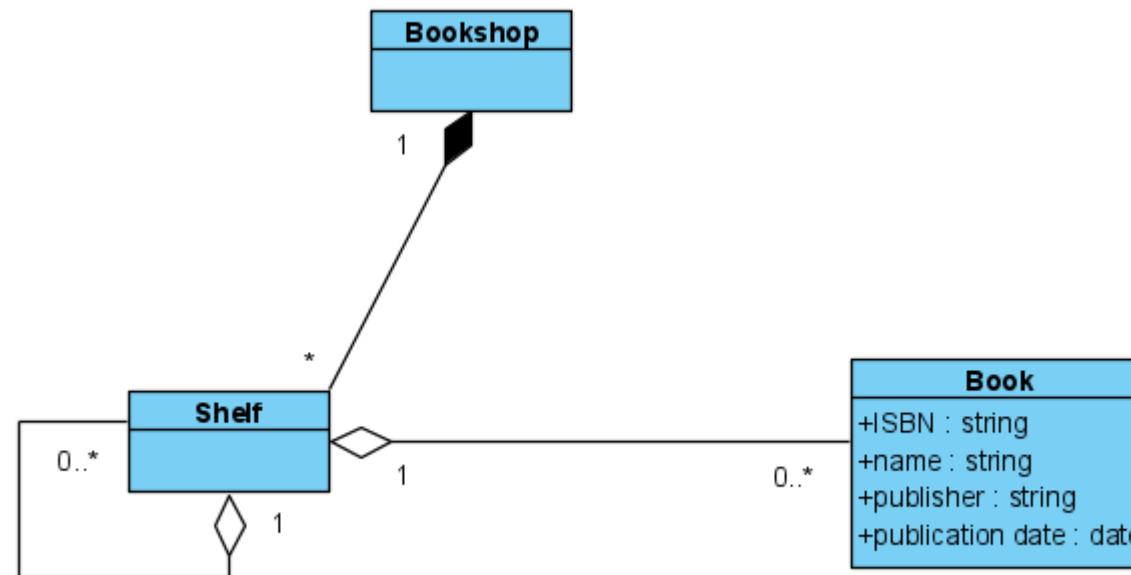
- In a bookshop there are several shelves which are arranged thematically,...



- In a bookshop there are several shelves which are arranged thematically, that is, **within the shelves there are shelves, within the shelves there are other shelves, and so on.**



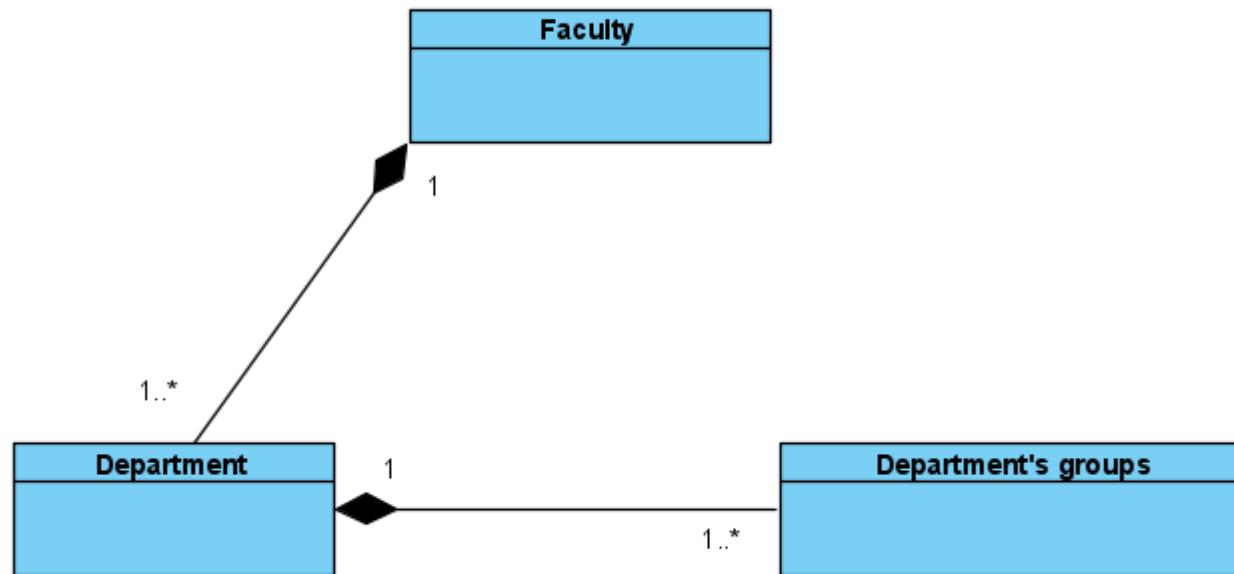
- On each shelf there are some books. When the bookstore is restructured, the books are distributed to other shelves. Each book has its own ISBN (string), title (string), publisher (string) and publication date (date).



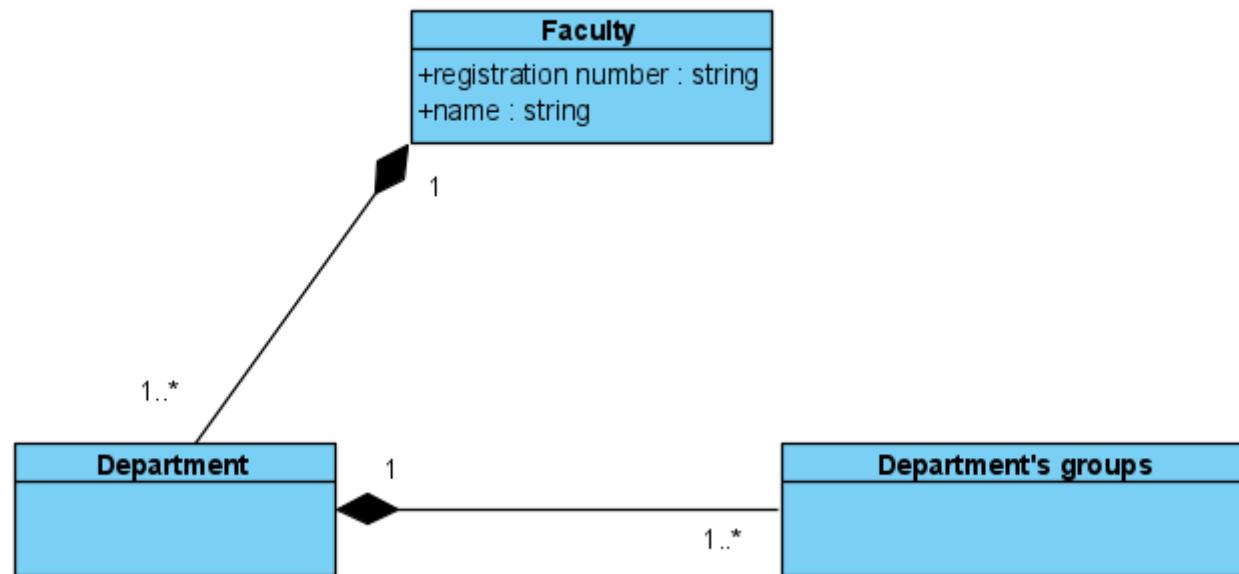
Example 2 [1]

- A faculty consists of one or more departments, and each department consists of one or more department's groups. The faculty has its own registration number (string) and name (string). The Department has its own name (string) and account number (string). The name and account number of the Department are also used by the department's groups. These groups also have their own group name (string) and additionally the name of the main laboratory (string). Make a class diagram.

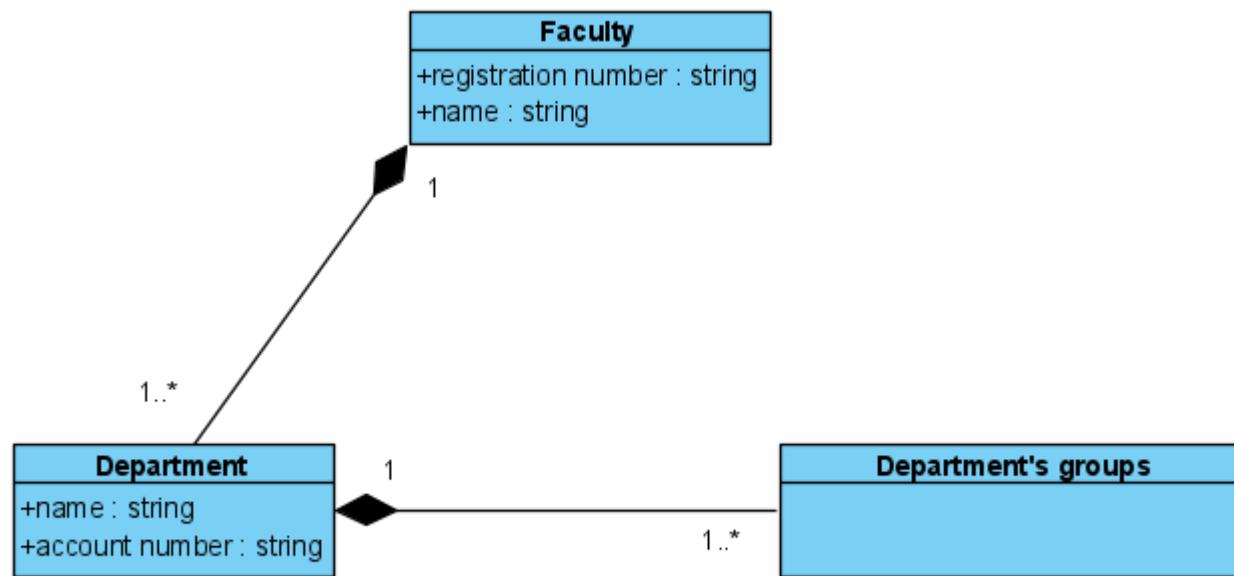
- A faculty consists of one or more departments, and each department consists of one or more department's groups.



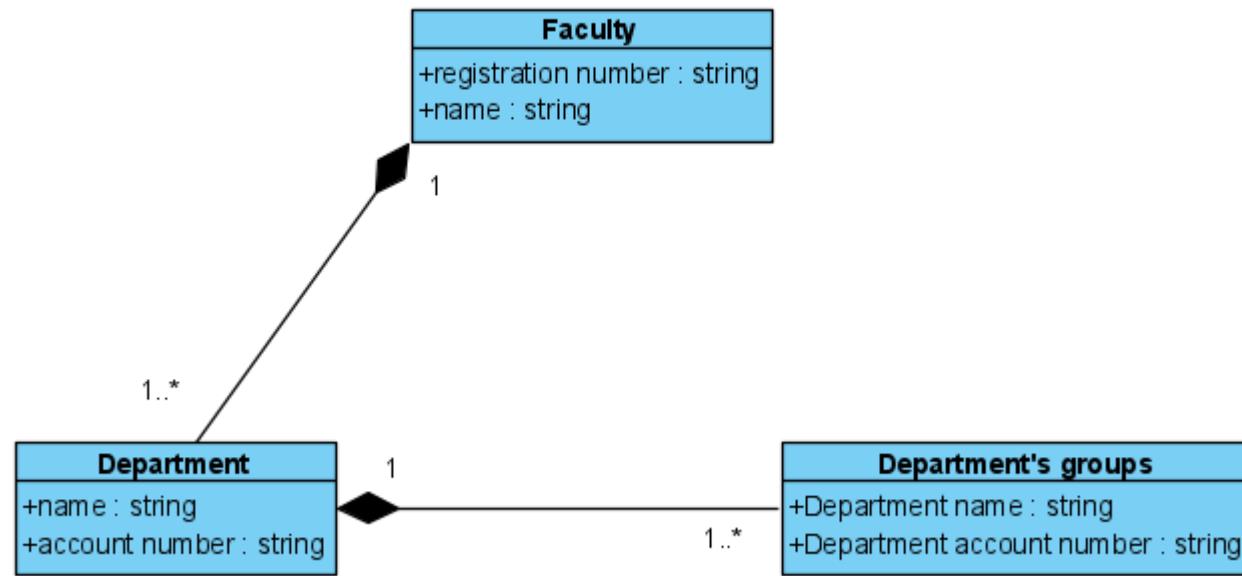
- The faculty has its own registration number (string) and name (string).



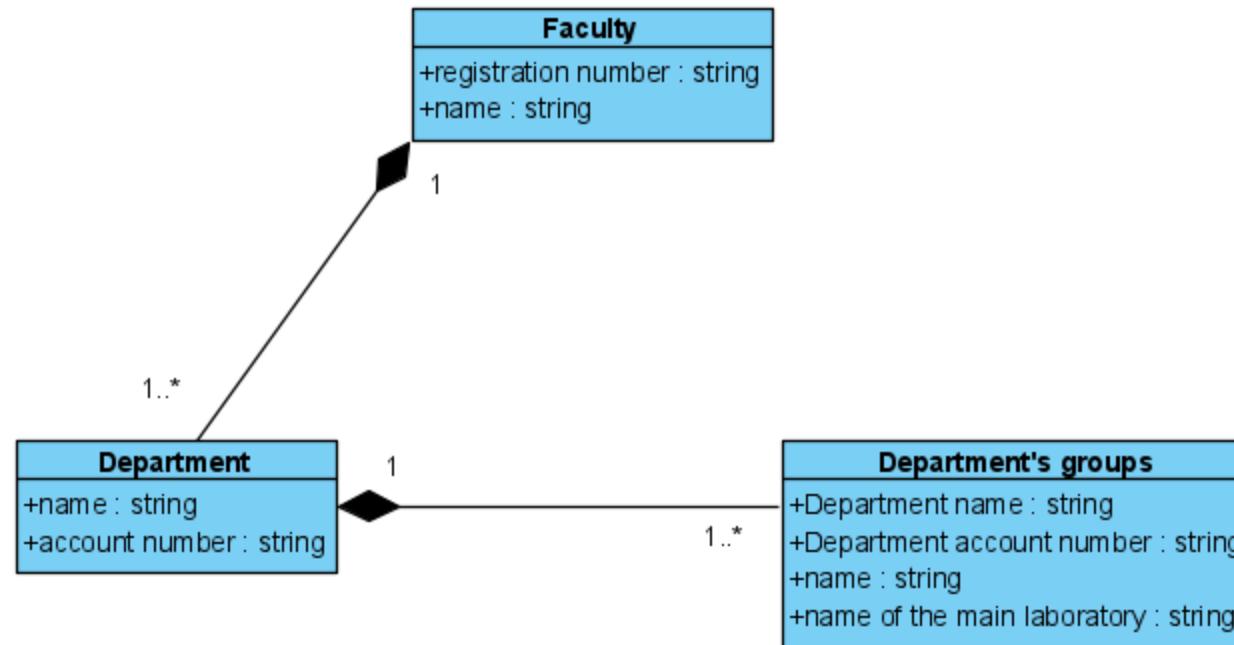
- The Department has its own name (string) and account number (string).



- The name and account number of the Department are also used by the department's groups.



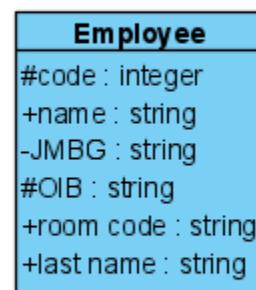
- These groups also have their own group name (string) and additionally the name of the main laboratory (string).



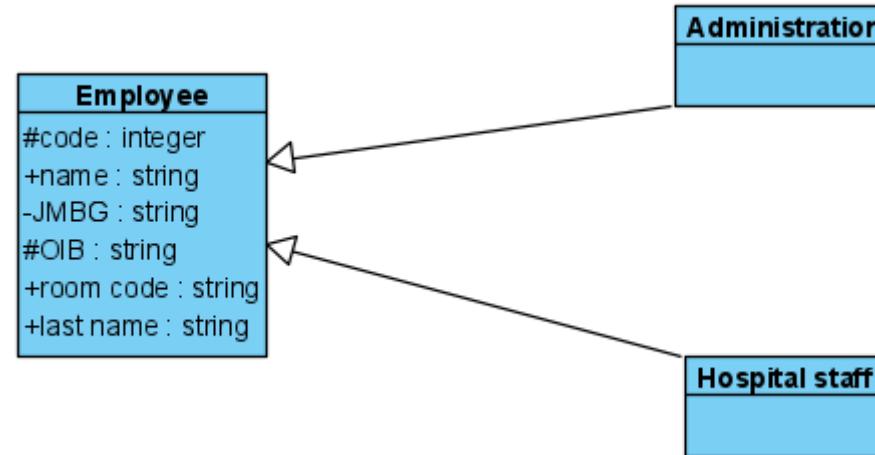
Example 3 [1]

- Some hospital information systems contain information about employees. Each employee has his own code (integer), name (string), last name (string), JMBG (string), OIB (string), and the room code in which he works (string). The code and OIB are protected data, the JMBG is private, and the person's first and last name are publicly available.
Nurses and doctors are medical staff. Medical staff and administration are different types of hospital staff. Staff members (employee) have a secure process to retrieve their code and OIB, and a public process to retrieve the room. Medical staff and administration staff can work in multiple rooms at once and have their own implementation of workroom retrieval. Doctors can make a diagnosis. Nurses can take a patient's temperature and blood pressure.

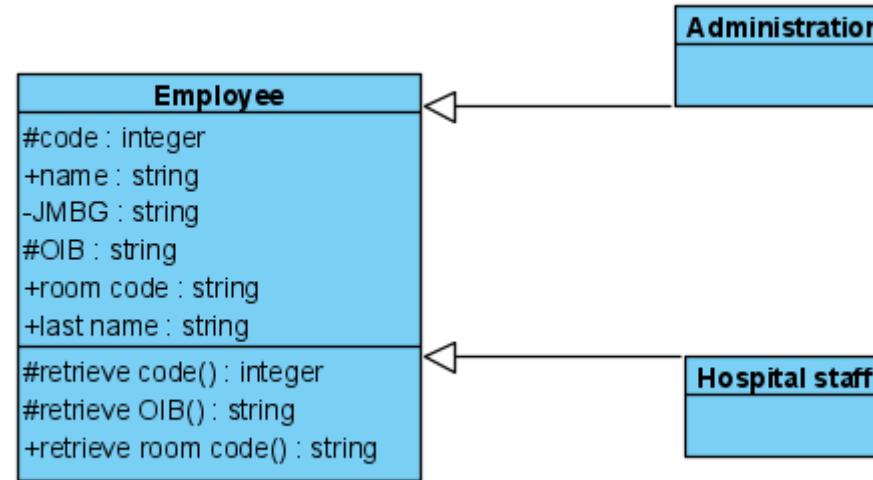
- Some hospital information systems contain information about employees. Each employee has his own code (integer), name (string), last name (string), JMBG (string), OIB (string), and the room code in which he works (string). The code and OIB are protected data, the JMBG is private, and the person's first and last name are publicly available.



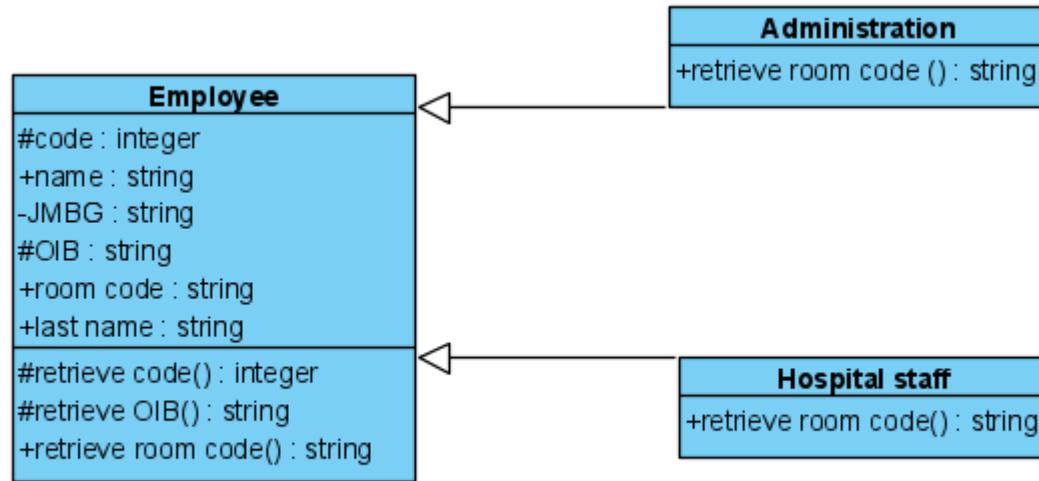
- Nurses and doctors are medical staff. Medical staff and administration are different types of hospital staff.



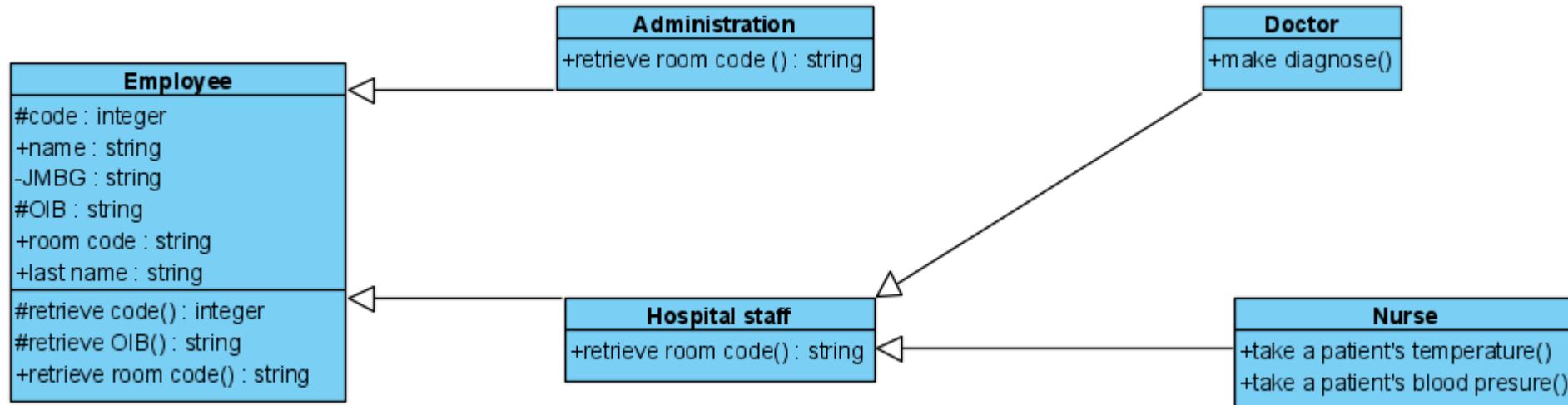
- Staff members (employee) have a secure process to retrieve their code and OIB, and a public process to retrieve the room.



- Medical staff and administration staff can work in multiple rooms at once and have their own implementation of workroom retrieval.



- Doctors can make a diagnosis. Nurses can take a patient's temperature and blood pressure.



Example 4 [2]

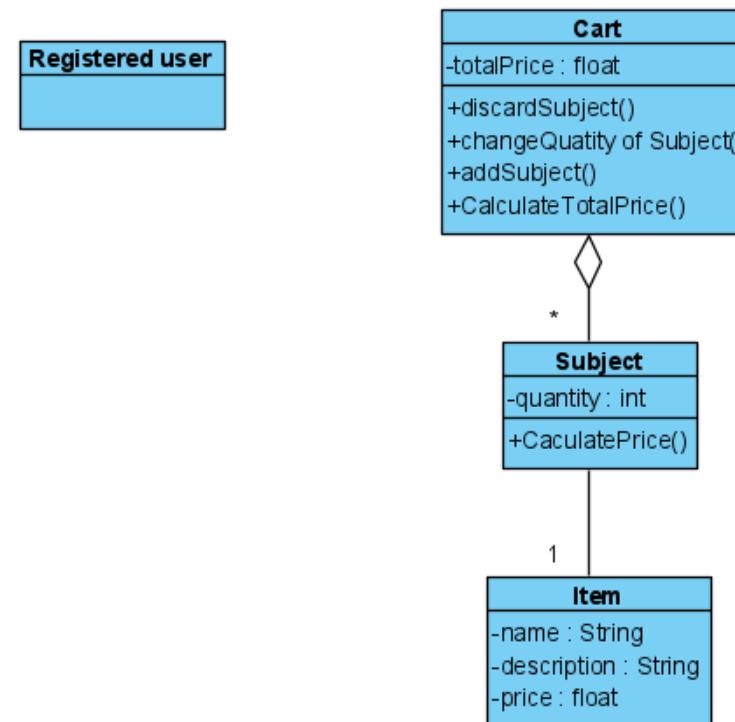
- Using class diagrams, model the order and payment system in the web store given by the following description.
- Registered users can shop in the web store. Users first add the desired items to the shopping cart by selecting the item and quantity and creating a subject in the shopping cart. For each item, the name, description and price per unit of quantity are given. In the shopping cart it is possible to add, discard or change the quantity of the inserted subject. You can always see the total price for all subjects in the shopping cart.
- When the registered user finishes adding all the items, he creates an order. When an order is created, the total amount to be paid is calculated and the date and time, delivery address, contact phone number and email address are recorded.
- When creating an order, the payment method is also selected: by bank card or cash (cash on delivery). When paying by card, it is necessary to enter the card details, and payment is made after confirming the order through an external payment service.
- For the order is monitored its status, which can be: "in progress" - before the order confirmation, "sent" - after confirmation until delivery, "delivered" - after successful delivery of the order.

- For registered users in the system, information about the contact phone number, e-mail address and shipping address is already available, and when creating an order, it is necessary to retrieve this information in advance and give the user the opportunity to correct it as desired. In addition, for registered users, all orders placed are stored and they can repeat them if they wish.
- The order can be canceled before it is confirmed, i.e. the user can cancel the order but the item remains in the shopping cart.

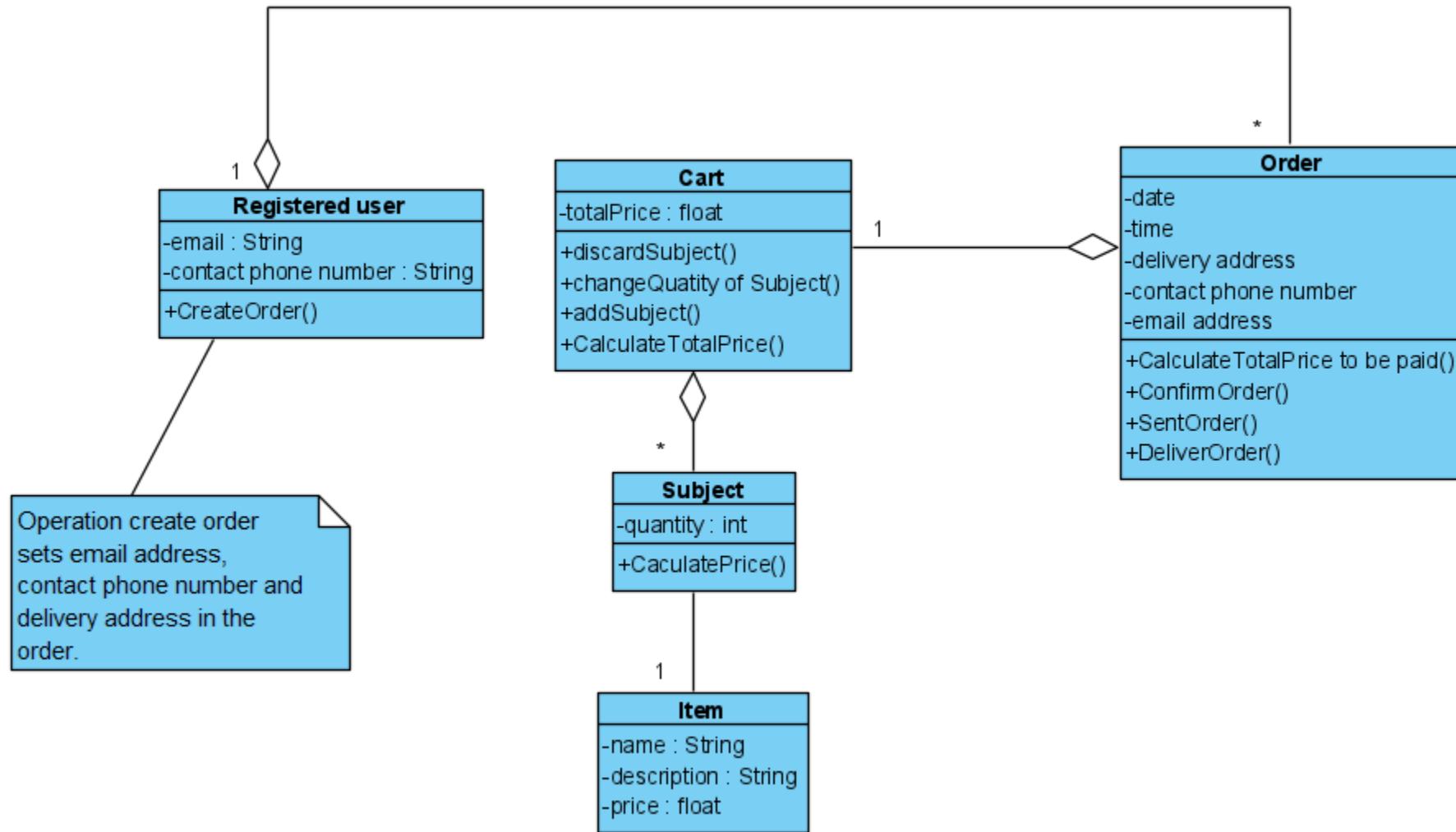
Classes

- Registered user
- Shopping Cart
- Subject
- Item
- Order
- Payment
- Bank card
- Cash
- Status
- address

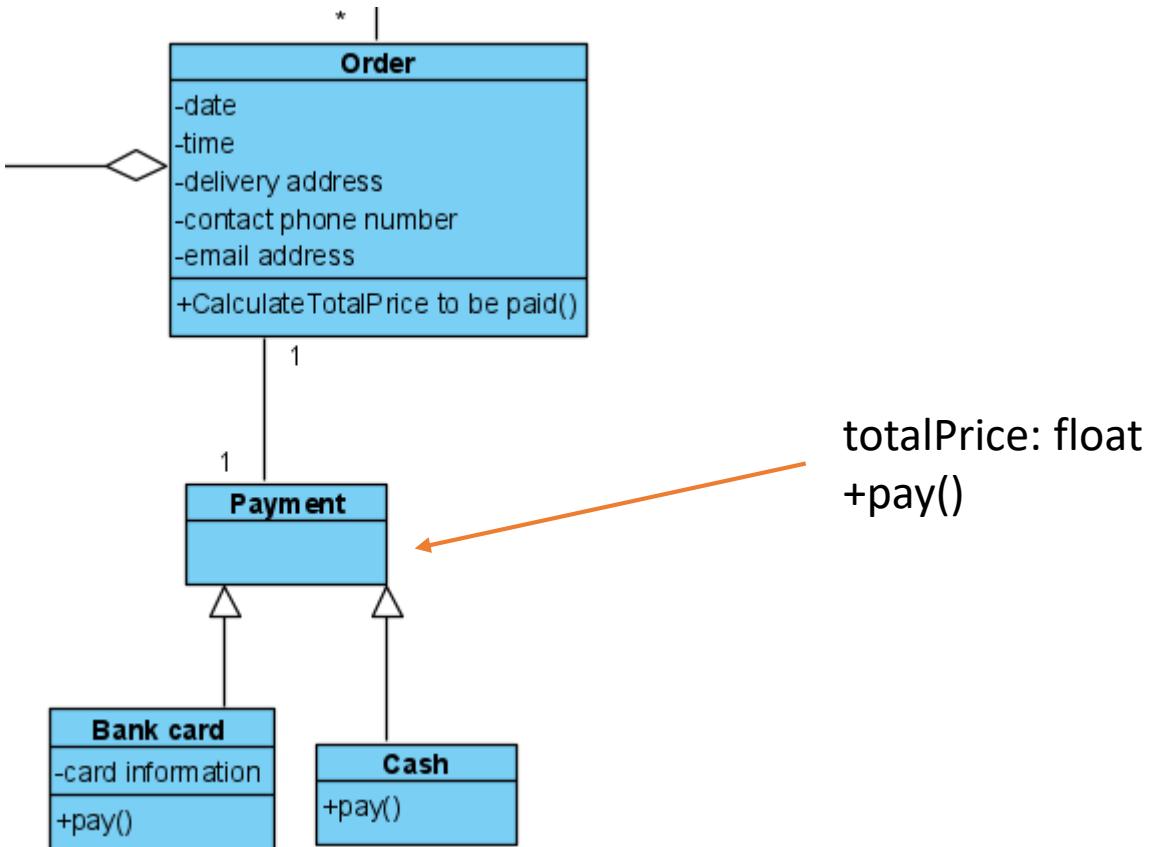
- Registered users can shop in the web store. Users first add the desired items to the shopping cart by selecting the item and quantity and creating a subject in the shopping cart. For each item, the name, description and price per unit of quantity are given. In the shopping cart it is possible to add, discard or change the quantity of the inserted subject. You can always see the total price for all subjects in the shopping cart.



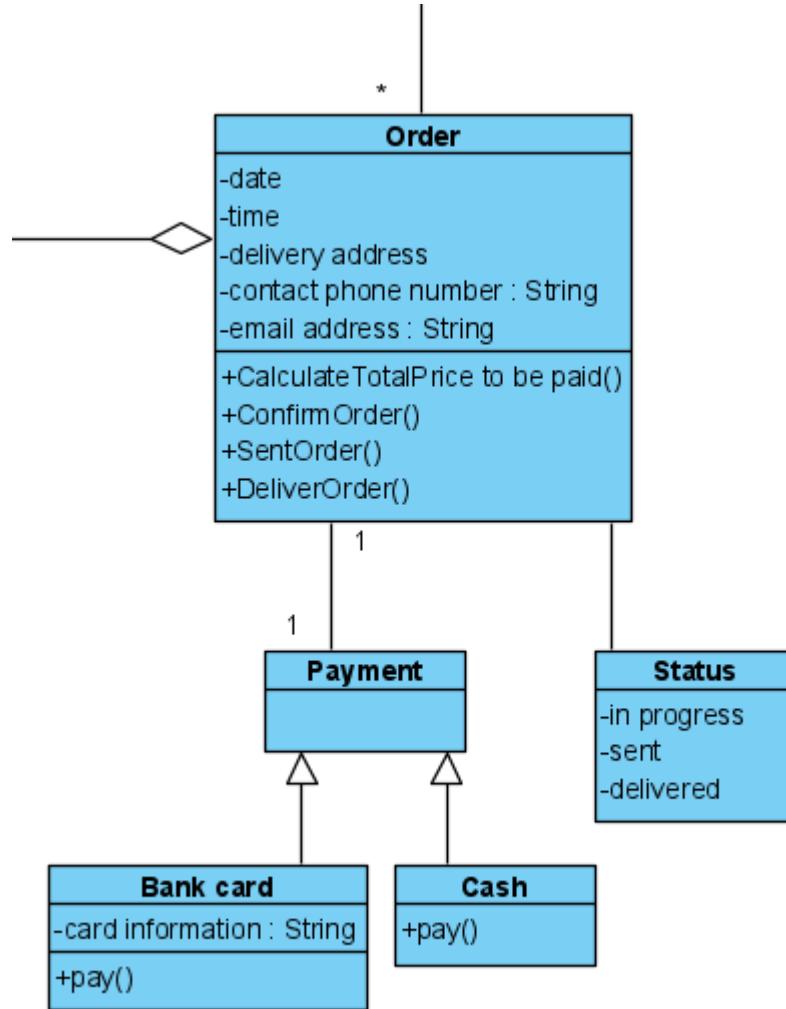
- When the registered user finishes adding all the items, he creates an order. When an order is created, the total amount to be paid is calculated and the date and time, delivery address, contact phone number and email address are recorded.
- Also, for registered users, all realized orders are remembered, and they can repeat them if desired.



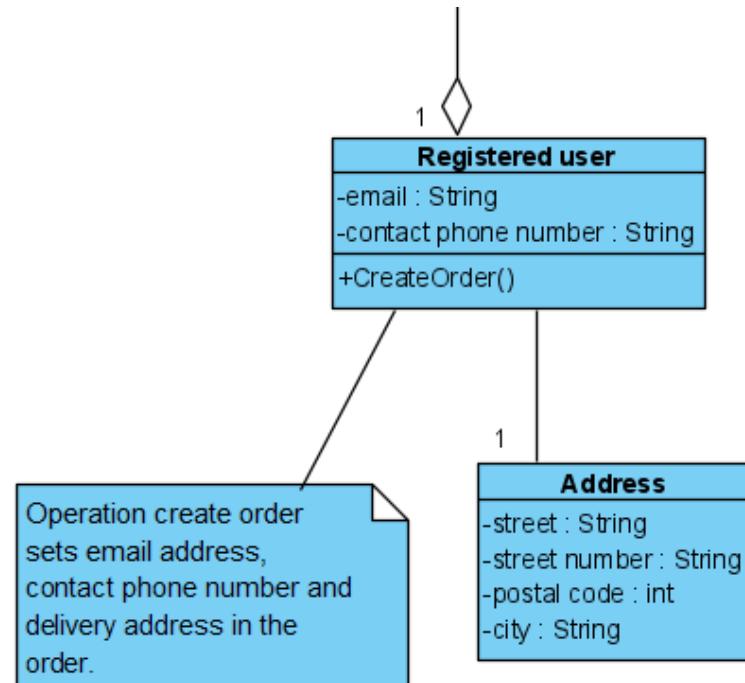
- When creating an order, the payment method is also selected: by bank card or cash (cash on delivery). When paying by card, it is necessary to enter the card details, and payment is made after confirming the order through an external payment service.



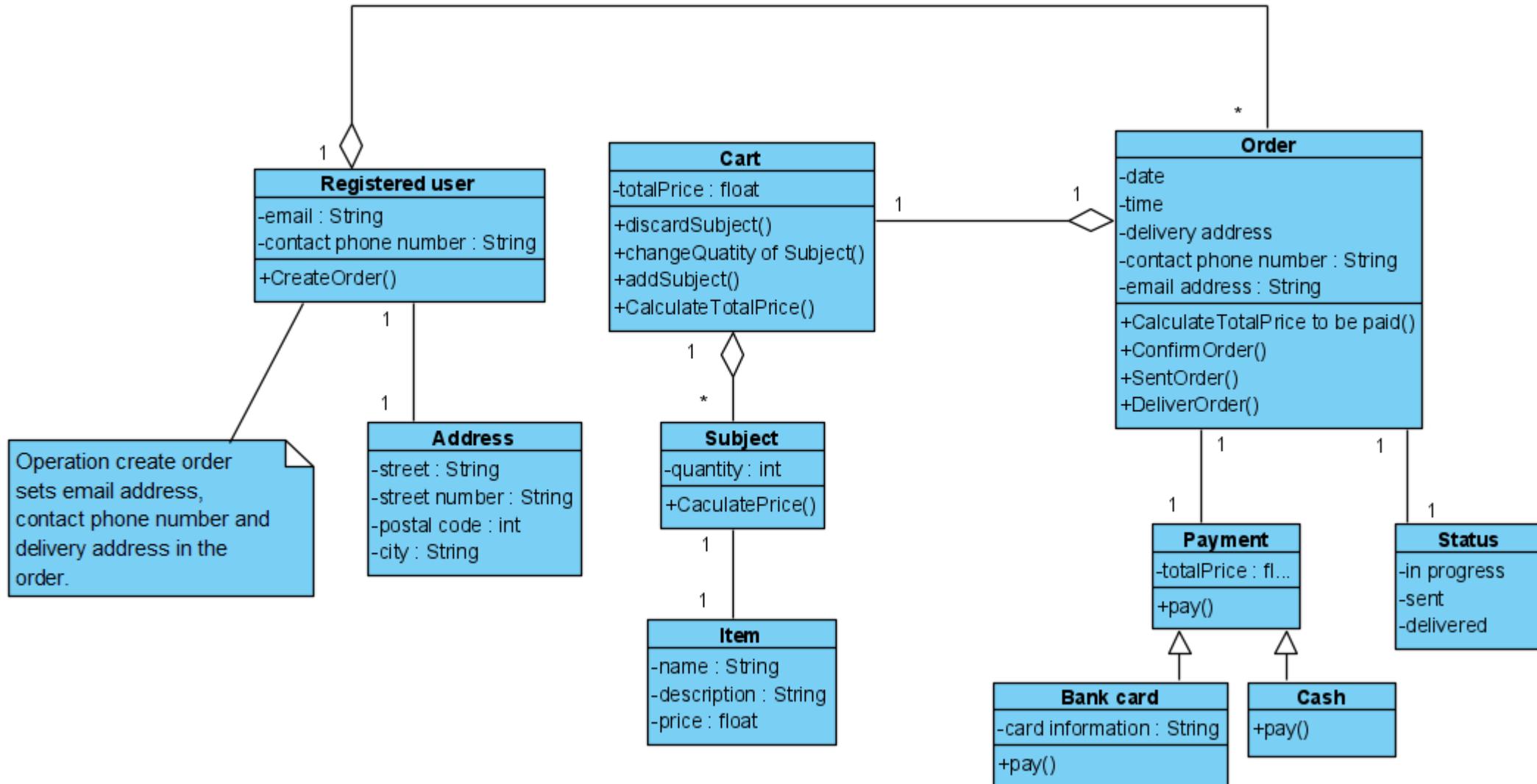
- For the order is monitored its status, which can be: "in progress" - before the order confirmation, "sent" - after confirmation until delivery, "delivered" - after successful delivery of the order.



- For registered users in the system, information about the contact phone number, e-mail address and shipping address is already available, and when creating an order, it is necessary to retrieve this information in advance and give the user the opportunity to correct it as desired. In addition, for registered users, all orders placed are stored and they can repeat them if they wish.



- The order can be deleted before it is confirmed, i.e. the user can cancel the order, but the subject still remain in the cart.
- There is no need to model a special withdrawal from the order, because the order will simply not be saved in that case.



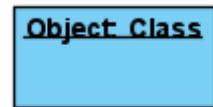
UML object diagrams

- **Object diagram** depicts instances (objects) of the classes present at certain time.

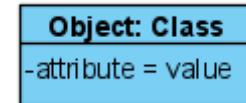
- Object diagrams are suitable for depicting class usage.
- They are showing “real world” objects and their relations.
- Needed when class diagrams are too abstract.
- Object diagrams shows an example of class usage in one time during SW operation.

Object diagrams notation

- **Object Names:** [5]
- Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon.



- **Object Attributes:** [5]
- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.

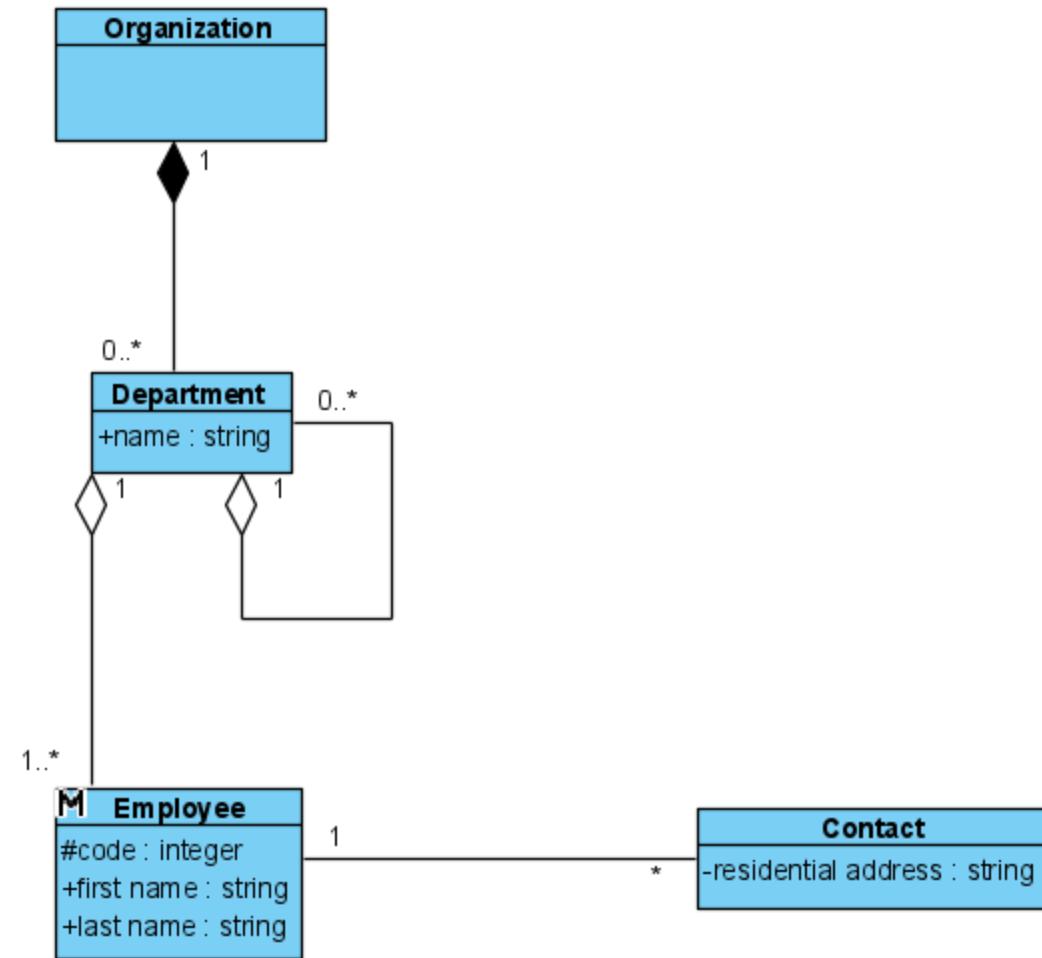


- **Links: [5]**
- Links tend to be instances associated with associations.

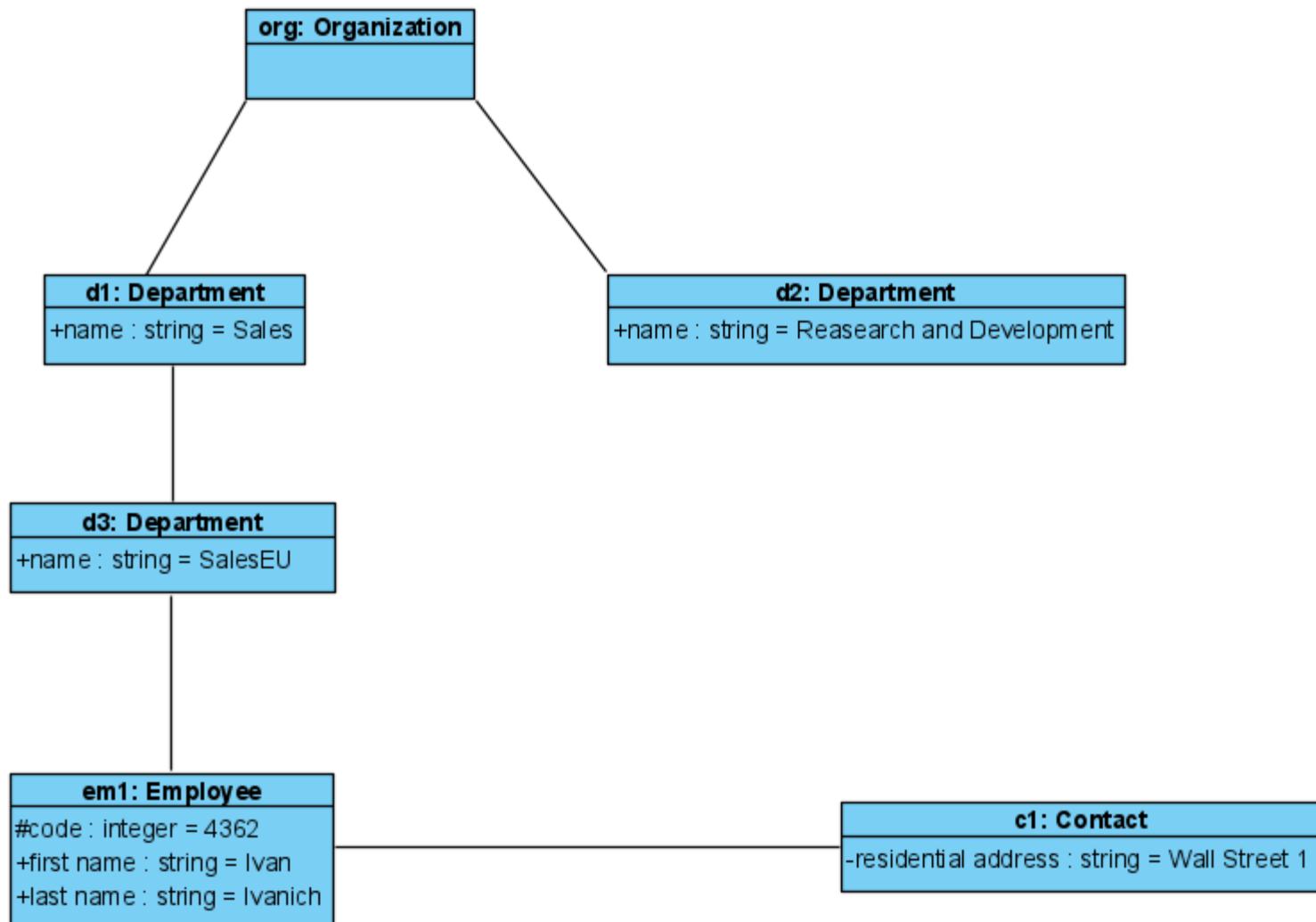


Example 5 [1]

- Some organizations have multiple departments. Each department has its own name (string). Departments can consist of sub-departments. Employees have a code (integer), a first and last name (string). The password is protected information. Some employees may have their own contact information associated with them (residential address).

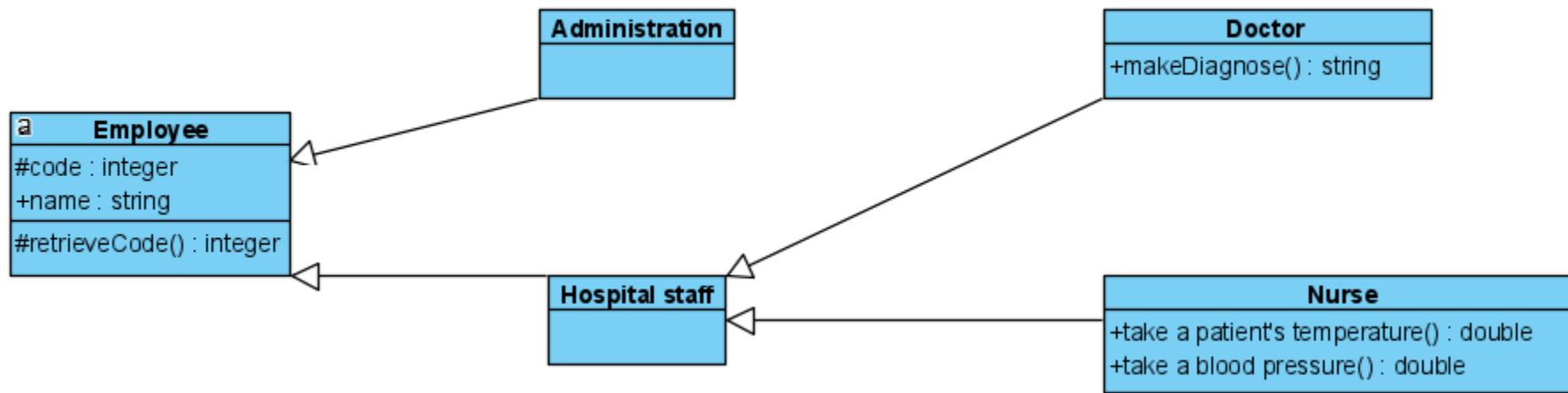


- Create an object diagram for the previous class diagram if there is an organization with two departments called "Sales" and "Research and Development". The "Sales" department has a subdepartment named " SalesEU". The SalesEU department has an employee with code 4362, name Ivan, last name Ivanich. His address is "WallStreet 1".



Example 6 [1]

- If the following class diagram is specified



- create an object diagram.

- The name of the nurse is Ivan Ivanich with code 1.

n1: Nurse
+code : integer = 1
+name : string = Ivan Ivanich
+take a patient's temperature() : double
+take a patient's blood pressure() : double
#retrieveCode() : integer

Homework assignment

- **Submission of specifications review reports**
- short opinion
- list of all comments, for bugs and for recommendations ,
- a summary: if it needs to be corrected or is good

Literature

- [1] Jović, A., Horvat, M. & Grudenić, I. (2014) UML-dijagrami: Zbirka primjera i riješenih zadataka. Zagreb. GRAPHIS d.o.o. Zagreb.
- [2] [https://www.fer.unizg.hr/ download/repository/UML zadaci za vjezbu - nadopuna sveucilisnog prirucnika\[1\].pdf](https://www.fer.unizg.hr/download/repository/UML_zadaci_za_vjezbu_nadopuna_sveucilisnog_prirucnika[1].pdf)
- [3] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/>
- [4] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>
- [5] <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/>

White-box testing

Structural testing (white-box testing)

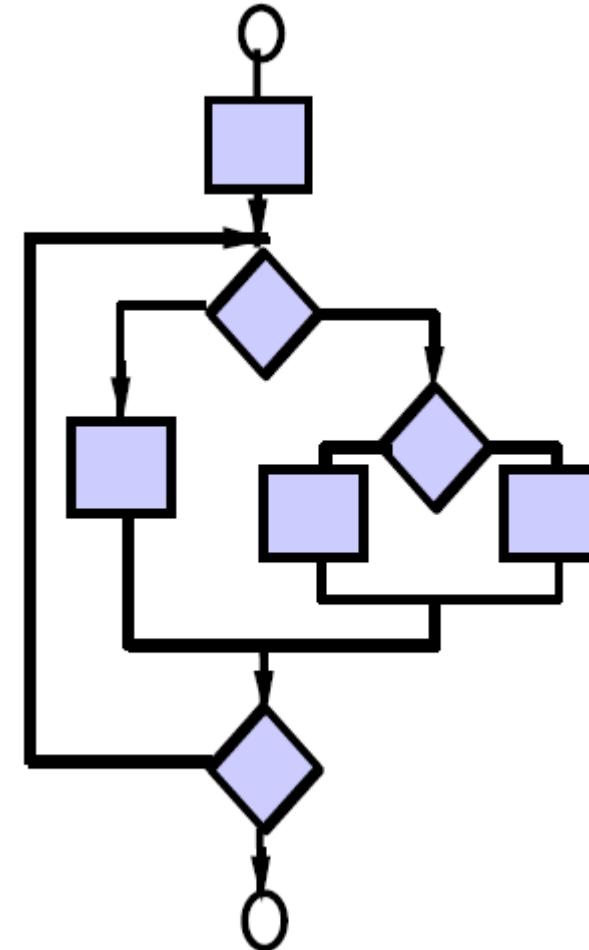
- Testing is based on knowledge of the internal structure of the program.

White - box

- The goal is to ensure that each condition has been satisfied at least once and every part of the source code has been executed at least once.

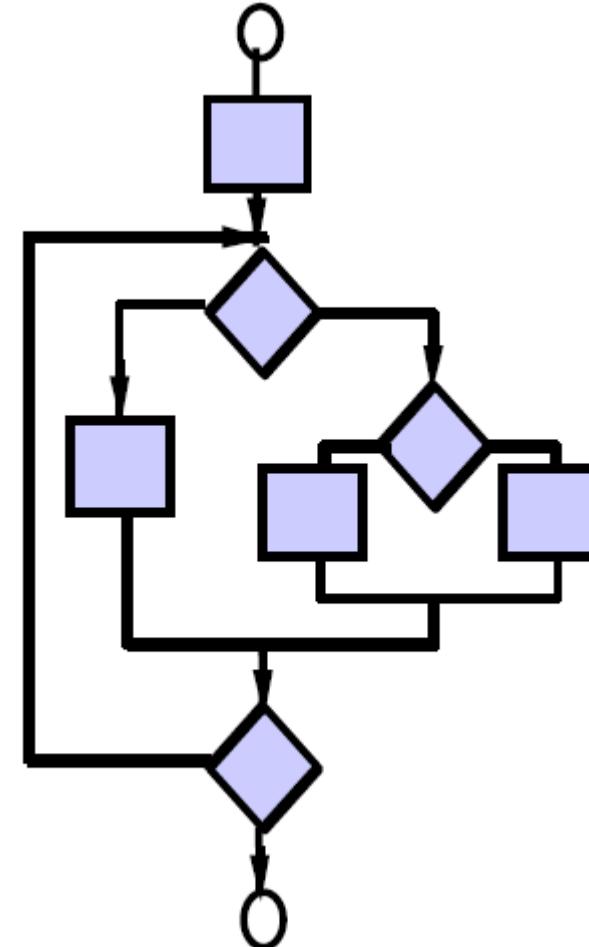
Basis Path Testing (1)

- A **basis path** is a path through a SW that contains at least one new piece of program code or a new condition.
 - At least one part of the path is not contained in other independent routes.
- The number of independent paths depends on the **cyclomatic complexity** - a measure of the logical complexity of the program.



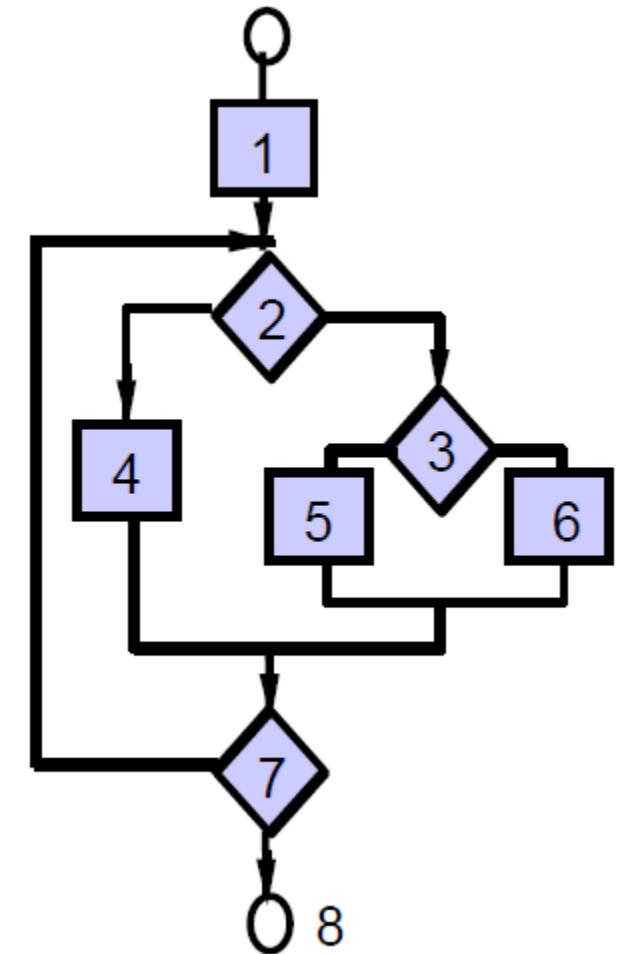
Basis Path Testing (2)

- Determining cyclomatic complexity $V(G)$ based on execution graph (activity diagram) G :
 - $V(G) = \text{number of simple decisions} + 1$
 - $V(G) = \text{number of contained areas of graph} + 1$
 - For the given example: $V(G) = 3 + 1 = 4$



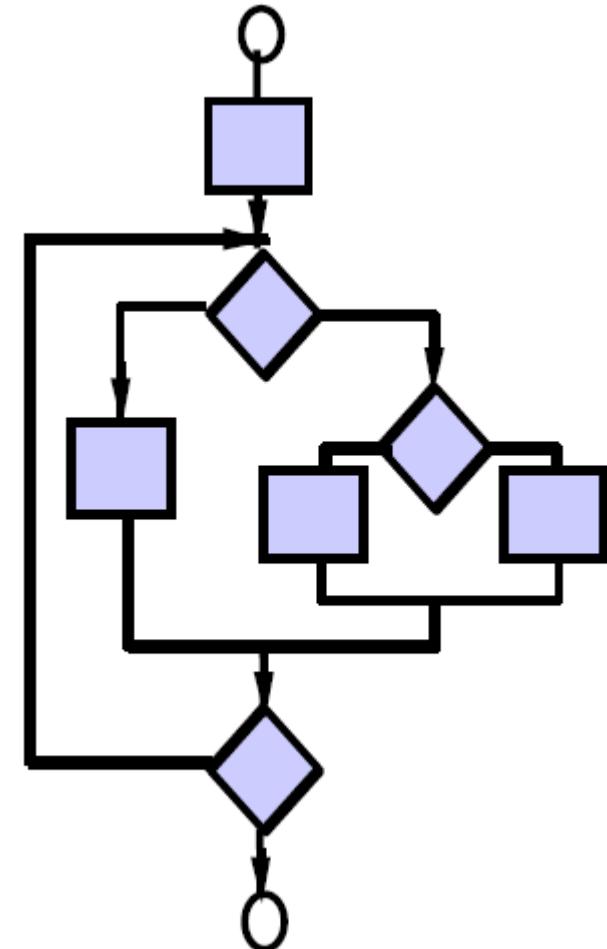
Basis Path Testing (3)

- Determination of independent paths. $V(G) = 4$; we can define 4 basic paths:
 - Path1: 1,2,4,7,8
 - Path2: 1,2,3,5,7,8
 - Path3: 1,2,3,6,7,8
 - Path4: 1,2,4,7,2,4,7,8
- Each additional path is a combination of existing routes and is therefore not independent. 1,2,3,6,7,2,4,7,8 -combination of paths 3&4
- Basic paths define test cases for structural testing.

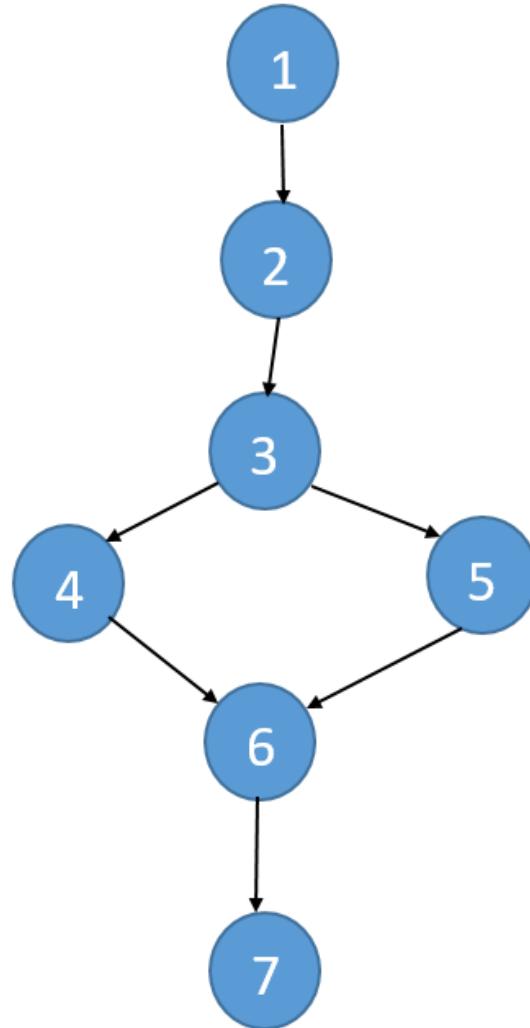


Basis Path Testing (4)

- Basic paths can be defined without using a graph, but graph makes it easier to follow the paths.
- Cyclomatic complexity can be determined by counting simple logical decisions. Complex decisions are considered two or more simple ones.
- Basic path testing is especially important for critical modules.

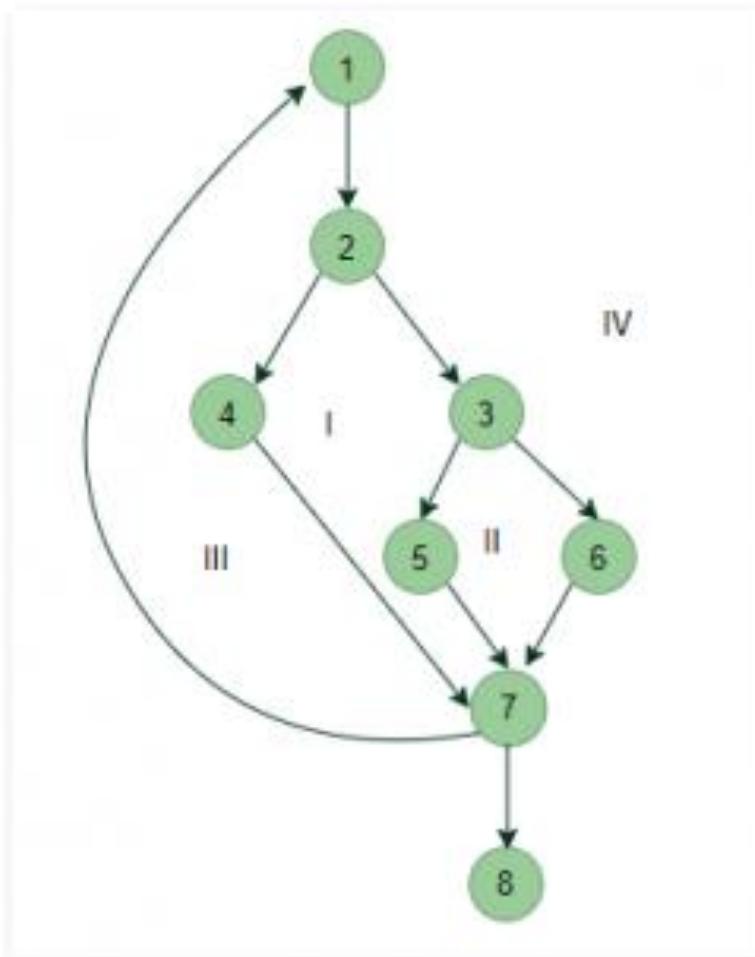


Example 1



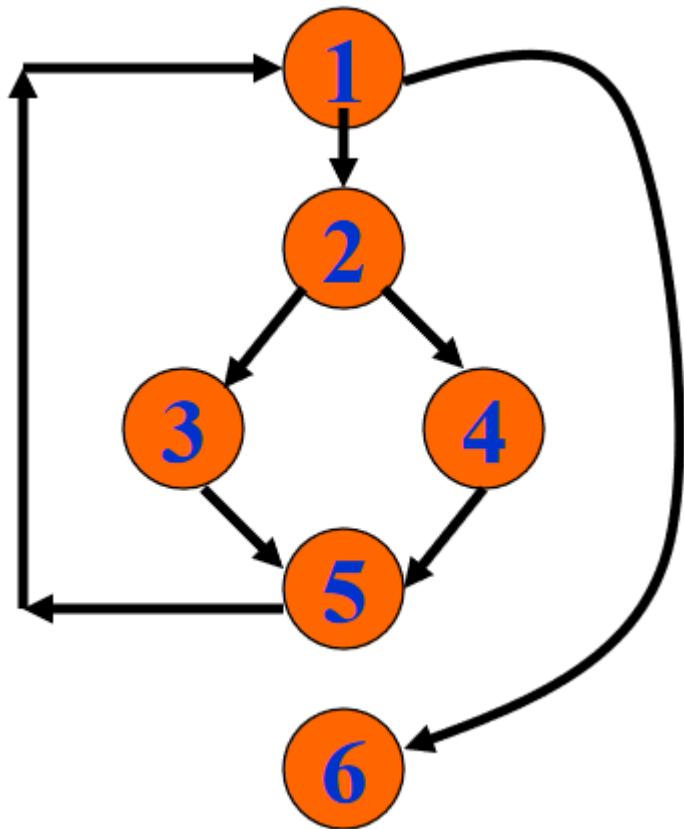
- $V(G) = 1 + 1 = 2;$
- Basic paths:
- Path 1: 1-2-3-4-6-7
- Path 2: 1-2-3-5-6-7

Example 2 [1]



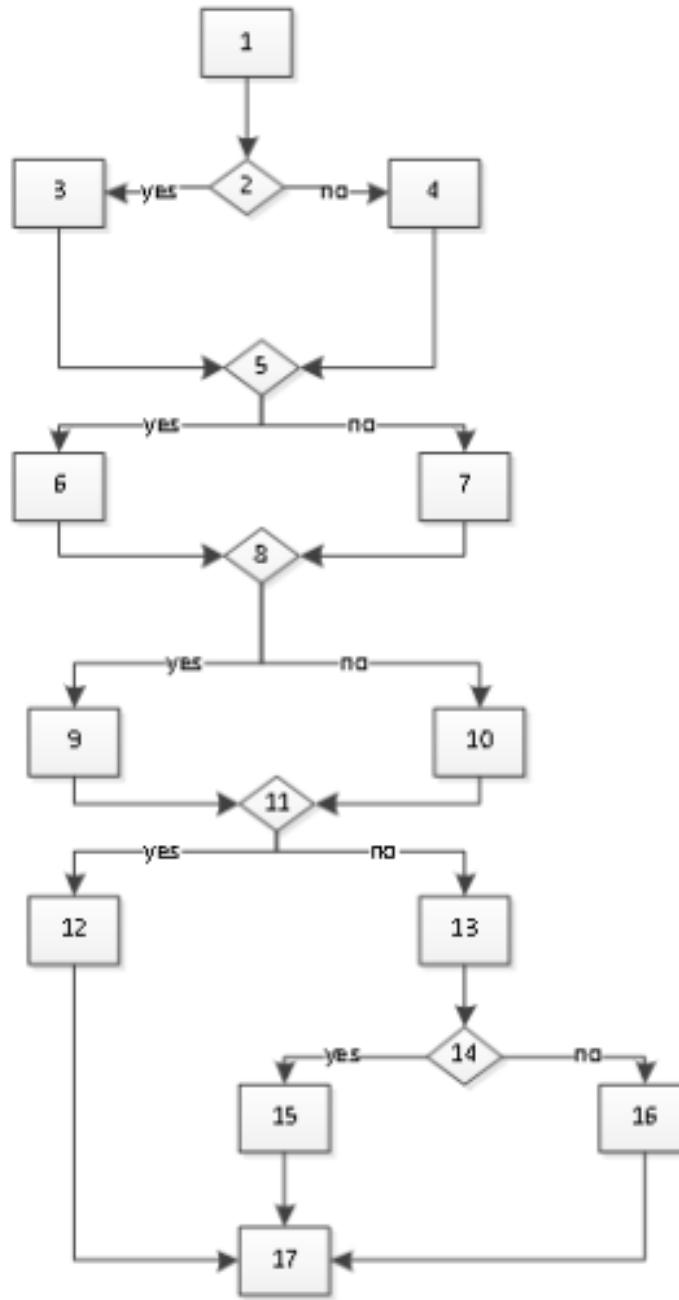
- $V(G) = 3+1 = 4$
- Basic paths:
- Path 1: 1-2-4-7-8
- Path 2: 1-2-3-5-7-8
- Path 3: 1-2-3-6-7-8
- Path 4: 1-2-4-7-1-2-4-7-8

Example 3 [2]



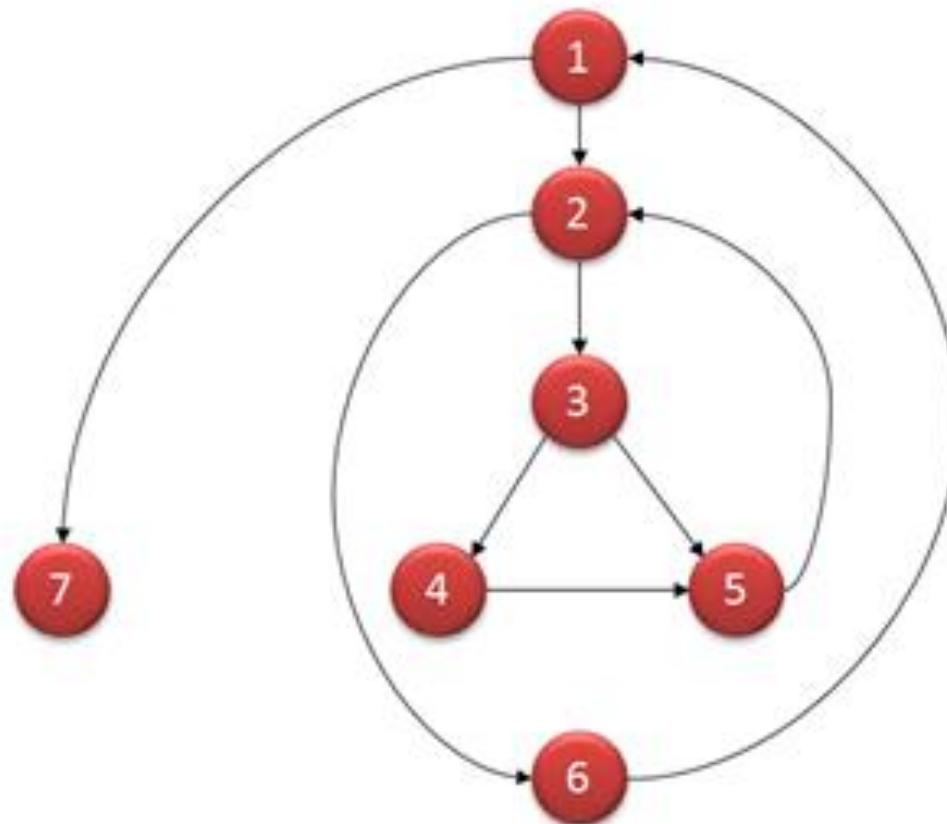
- $V(G) = 2+1=3$
- Basic paths:
- Path 1: 1-6
- Path 2: 1-2-3-5-1-6
- Path 3: 1-2-4-5-1-6

Example 4 [3]



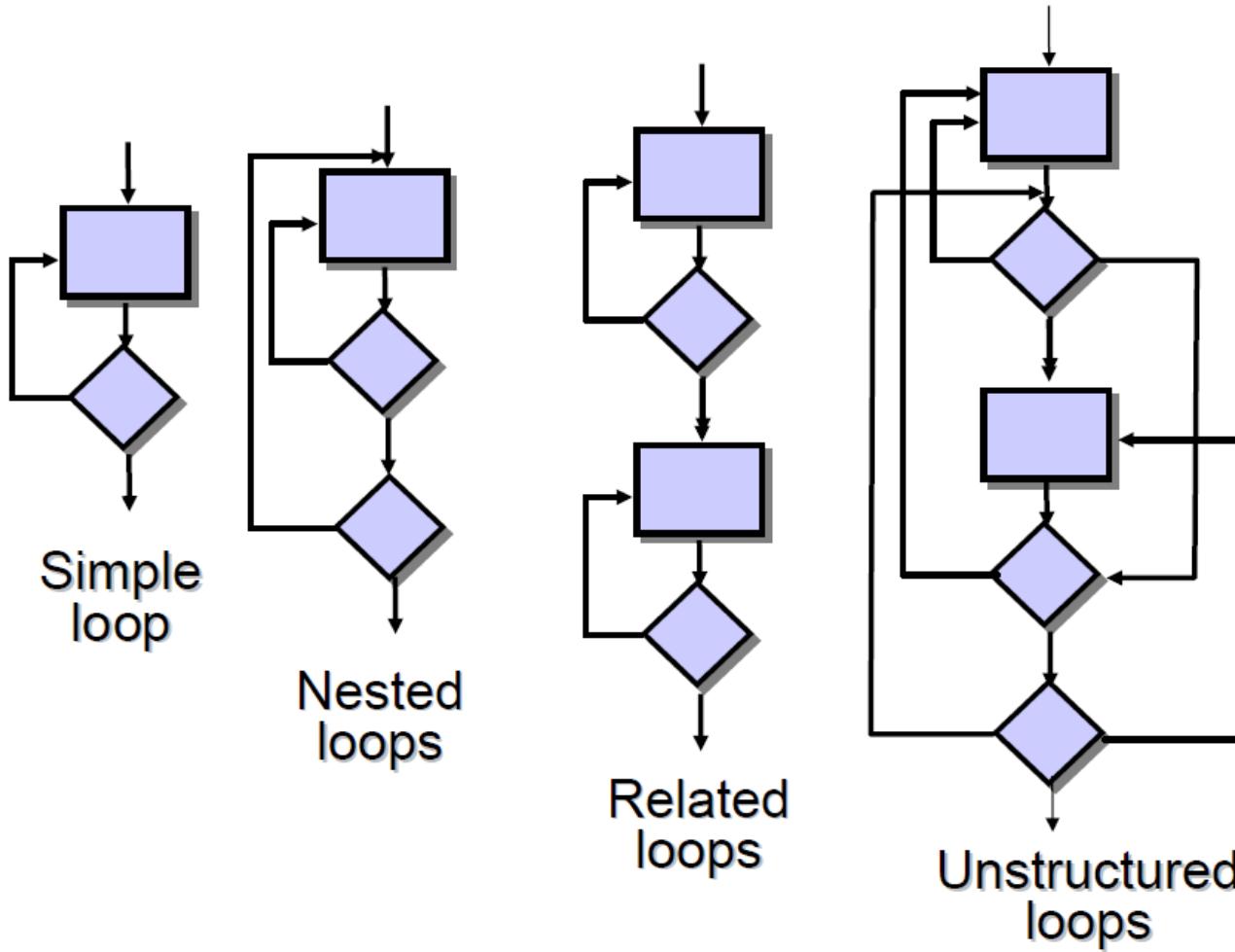
- $V(G) = 5+1 = 6$
- Basic paths:
- Path 1: 1-2-3-5-6-8-9-11-12-17
- Path 2: 1-2-4-5-6-8-9-11-12-17
- Path 3: 1-2-3-5-7-8-9-11-12-17
- Path 4: 1-2-3-5-6-8-10-11-12-17
- Path 5: 1-2-3-5-6-8-9-11-13-14-15-17
- Path 6: 1-2-3-5-6-8-9-11-13-14-16-17

Example 5 [4]



- $V(G) = 3 + 1 = 4$
- Basic paths:
- Path 1: 1-7
- Path 2: 1-2-6-1-7
- Path 3: 1-2-3-4-5-2-6-1-7
- Path 4: 1-2-3-5-2-6-1-7

Testing of loops



Testing of simple loops

- A set of test cases should include: Leaving out the loop
 - A single pass through the loop
 - Two passes through the loop
 - m passes through a loop where $m < n$
 - $n-1, n, n+1$ passes through the loop
- n - the maximum number of loop passes allowed

Testing of nested loops

- Start with the innermost loop. All other loops should be executed with the least number of iterations.
- Test the loop for $\min + 1$, $\max - 1$ and the typical number of iterations.
- Move to one level a more external loop and test it the same way. Continue this way until you have completed the outermost loop.

Testing of related loops

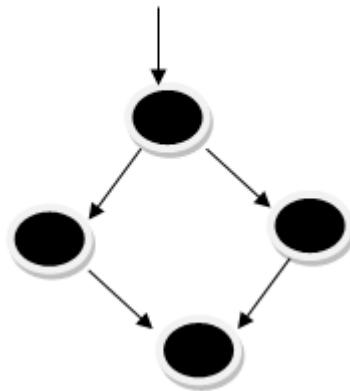
- If the loops are independent, then treat each one as a simple loop
- If the loops are not independent, treat them as nested loops.
 - Dependency example: The final counter of one loop affects the initialization of the next loop.

Testing of conditions

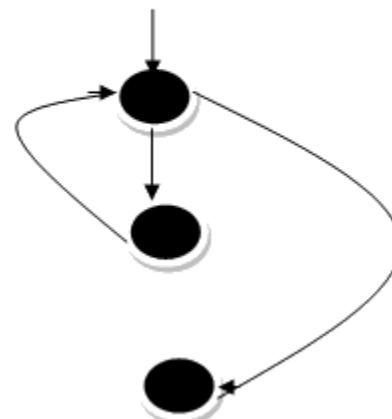
- Each decision is made based on some condition.
- Condition testing is based on the selection of such test cases to verify the correctness of the decisions against the decision parameters.
- The following errors are possible:
 - Logical operator error (inappropriate, missing, redundant)
 - Logical variable error
 - Error in brackets of logical expression
 - Error in relational operator ($>$, $<$, \leq , \geq , $=$, \neq , $!=$)
 - Error in arithmetic operator
- The test cases for testing the conditions partially overlap with the test cases for testing the basic paths.

Conditions, Loops -> Graphs [6]

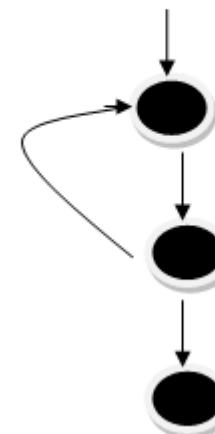
If-then-else:



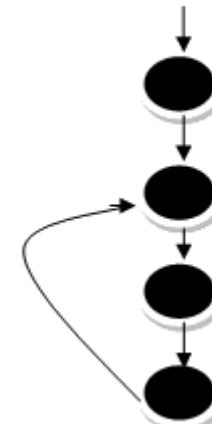
While:



Do-While:



For:



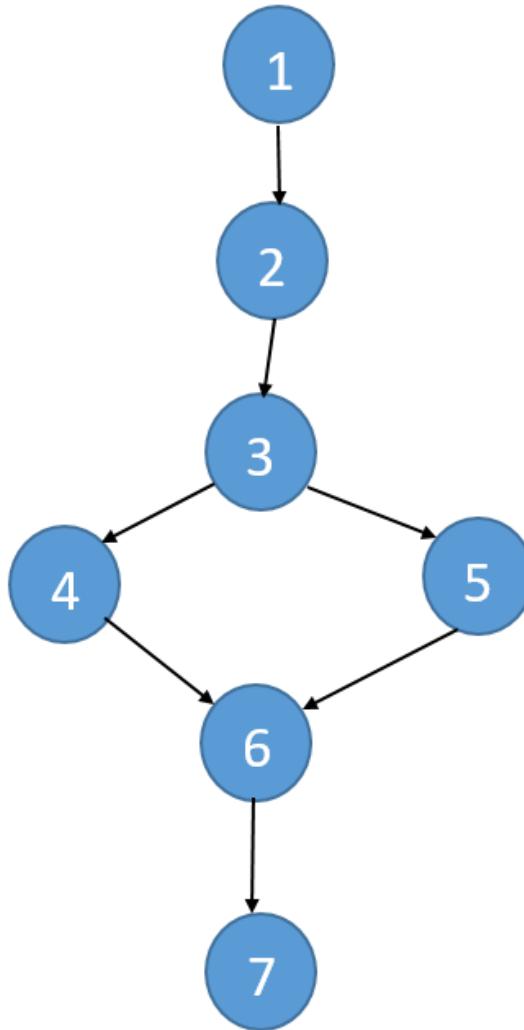
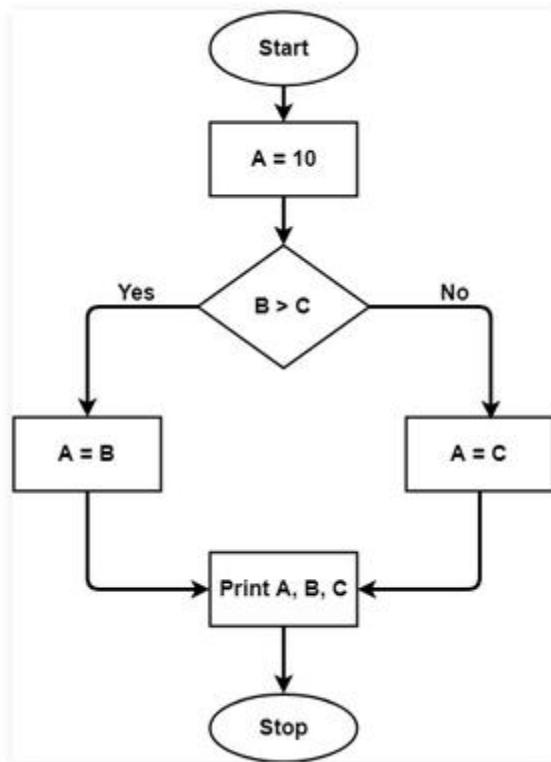
Example 6 [5]

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths. Using the basic paths, please define test cases that allow us to traverse each of the basic paths.

```
%initialization
prompt_1='Enter B: ';
prompt_2='Enter C: ';
B=input(prompt_1);
C=input(prompt_2);
%
A=10;

%
if B>C
    A=B;
else
    A=C;
end
%
sprintf('A= %.2f',A)
sprintf('B= %.2f',B)
sprintf('C= %.2f',C)
```

[5]

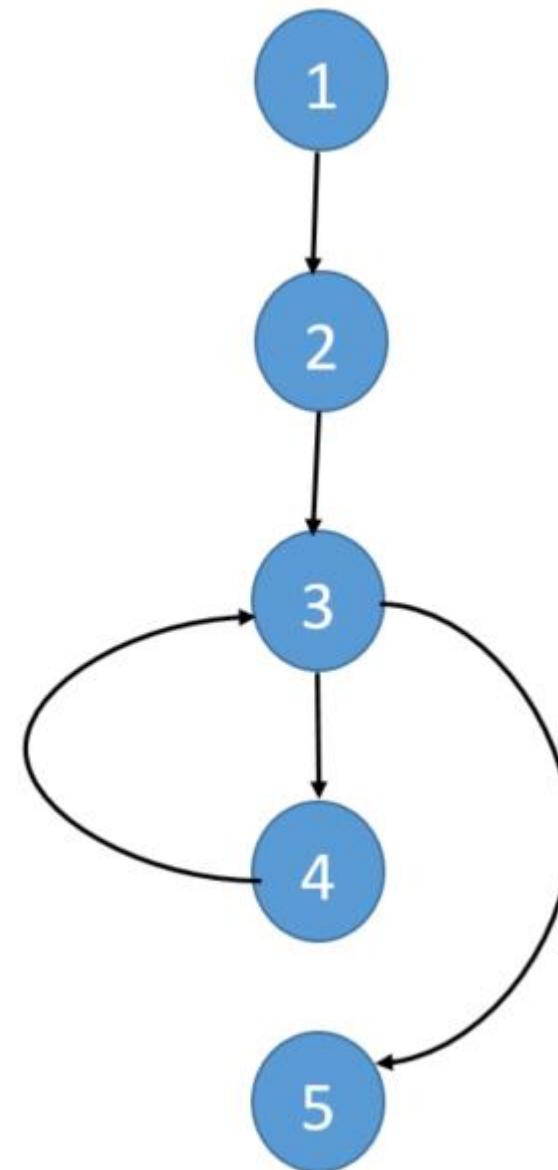
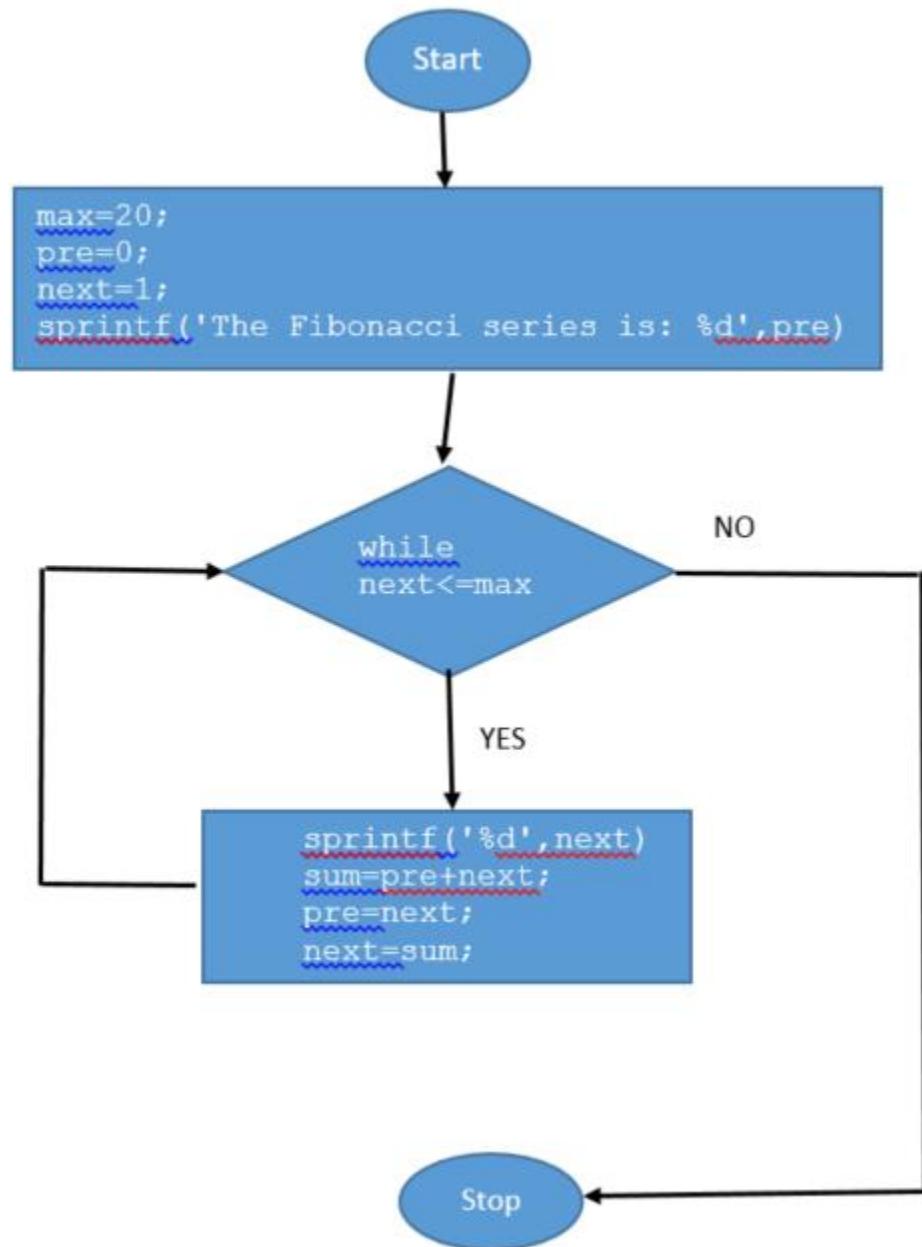


- $V(G) = 1 + 1 = 2$;
- Basic paths:
- Path 1: 1-2-3-4-6-7
- Path 2: 1-2-3-5-6-7
- Test cases:
 - Path 1:
 - Input parameters: B=13, C=11, A=10
 - Output parameters: B=13.00, C=11.00, A=13.00
 - Path 2:
 - Input parameters: B=15, C=21, A=10
 - Output parameters: B=15.00, C=21.00, A=21.00

Example 7 [6]

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths. Using the basic paths, please define test cases that allow us to traverse each of the basic paths.

```
%% Example 7 code Fibonacci series
%initialization
max=20;
pre=0;
next=1;
sprintf('The Fibonacci series is: %d',pre)
%
while next<=max
    sprintf('%d',next)
    sum=pre+next;
    pre=next;
    next=sum;
end
```



- $V(G) = 1 + 1 = 2$;
 - Basic paths:
 - Path 1: 1-2-3-5
 - Path 2: 1-2-3-4-3-5
-
- Test cases:
 - Path 1:
 - Input parameters: next=21; pre=13; max=20;
 - Output parameters:
 - 'The Fibonacci series is 13'
 - next=21; pre=13; max=20;
 - Path 2:
 - Input parameters: next=8; pre=5; max=20;
 - Output parameters :
 - 'The Fibonacci series is 5.'
 - 8, 13
 - next=21; pre=13; max=20

Homework assignment

- System Design (individual):
 - Draw a UML component diagram or a UML deployment diagram for your project idea.
 - For each connection/link between components, write what it is used for (information flow, requirements,... what information is transferred via each connection between components)

Literature

- [1] <https://www.geeksforgeeks.org/software-engineering-white-box-testing/>
- [2] https://www.spu.edu.sy/downloads/files/1559413788_Session05.pdf
- [3] <https://liveronika.wordpress.com/2013/12/09/white-box-testing-with-cyclomatic-complexity-graph/>
- [4] <https://medium.com/@danielsolano/white-box-test-cyclomatic-complexity-276fb5803a04>
- [5] <https://www.geeksforgeeks.org/cyclomatic-complexity/>
- [6] <https://www.educba.com/cyclomatic-complexity/>

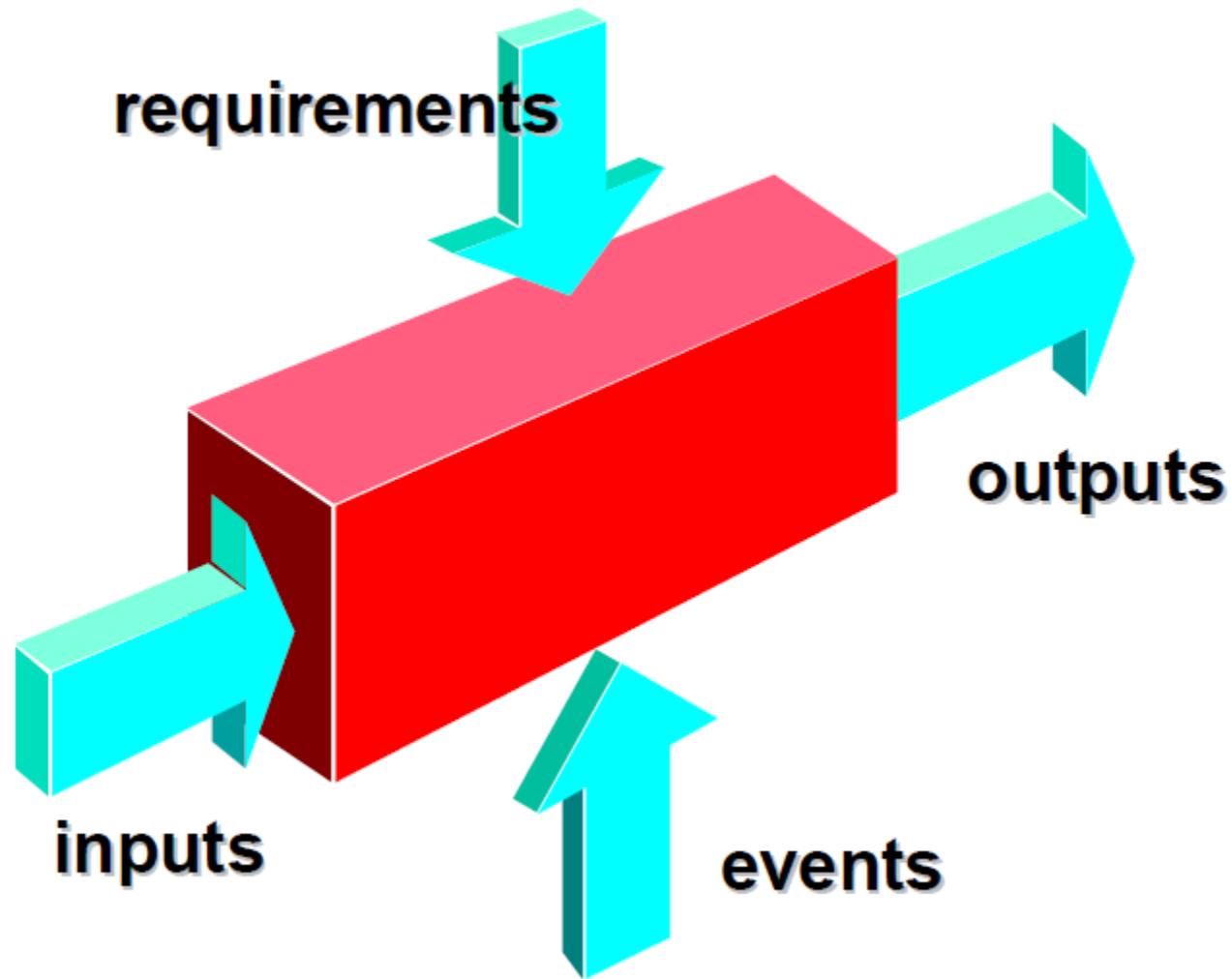
Black box testing

Black box testing definition

- Behavioral testing(black-box testing)
- Testing is based on the (declared) external properties of the system.

Behavioral testing (black-box)

- Unlike white-box, black-box testing is used in the later stages of the testing process.
- It does not focus on the control flow but the processed information.
- Test case planning is based on the following questions:
 - How to verify functional correctness?
 - How to check system behavior and throughput?
 - What input form good test cases?
 - Is the system sensitive to certain input values?
 - What are the boundaries of data classes?
 - What range of data can the system tolerate?
 - What impact do specific combinations of data have on the performance of the system?



Equivalence partitioning

- Equivalence partitioning is a procedure for determining the classes of input parameters from which to select test data.
- An equivalence class is a group of valid or invalid input conditions.
- Depending on the input conditions, we define the equivalent classes as follows:
 - The input condition is field-specified - specifies one valid and two invalid equivalent classes.
 - The input condition is a specific value - it specifies one valid and two invalid equivalent classes.
 - The input condition is determined by the membership of the set - it specifies one valid and one invalid equivalent class.
 - Boolean - Specifies one valid and one invalid equivalent class.

Boundary value analysis

- Boundary value analysis – (BVA) leads to the selection of test cases based on boundaries of input data.
 - Eg: the input condition is determined by the region bounded by a and b, - test cases should use the values a and b and the lower and higher values.
 - ...
- Testing with a BVA is not only about the input but also the output!

Other behavior testing techniques

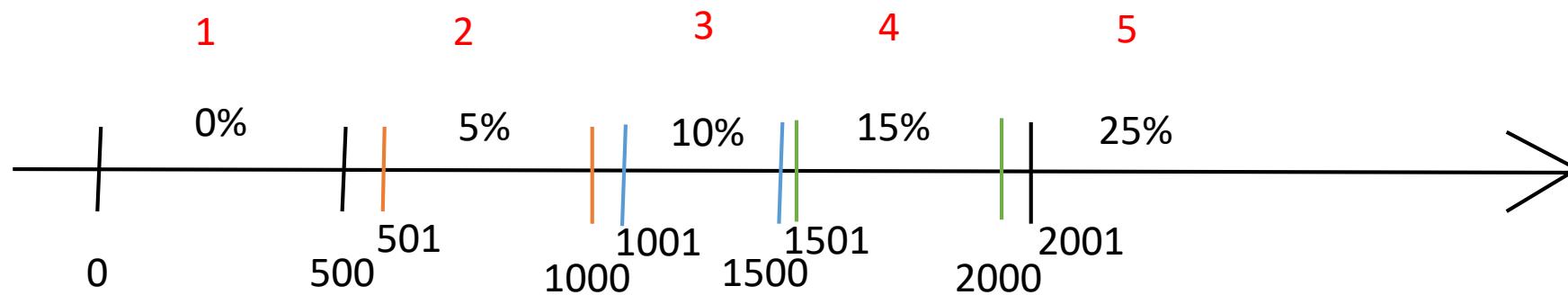
- Error guessing methods
- Decision table techniques
- Cause-effect graphing

Equivalence partitioning

Example 1 [1]

- Purchase discount is 0% for up to 500 US\$, 5% is added for each additional 500 US\$ up to 2000 US\$, and 25% is applied for above 2000 US\$. Which test inputs in US\$ would be selected for valid equivalence partitions?
 - a) 250, 700, 1400, 1800, 4000
 - b) 250, 1400, 3000
 - c) -100, 250, 650, 1300, 1700, 2900
 - d) 200, 720, 1600, 1800, 2100

Purchase discount is 0% for up to 500 US\$, 5% is added for each additional 500 US\$ up to 2000 US\$, and 25% is applied for above 2000 US\$. Which test inputs in US\$ would be selected for **valid** equivalence partitions?



- a) 250, 700, 1400, 1800, 4000
- b) ~~250, 1400, 3000~~
- c) ~~100, 250, 650, 1300, 1700, 2900~~
- d) 200, 720, ~~1600, 1800~~, 2100

Example 2 [2]

- One of the fields on a form contains a text box that accepts numeric values in the range of 18 to 25. Identify the invalid Equivalence class.
- a) 17
- b) 19
- c) 24
- d) 21

Solution:

- The text box accepts numeric values in the range of 18 to 25 (18 and 25 are also part of the class). So this class becomes our valid class. But the question is to identify invalid equivalence classes. The classes will be as follows:

Class I: values < 18 => invalid class

Class II: 18 to 25 => valid class

Class III: values > 25 => invalid class

- 17 falls under an invalid class. 19, 24 and 21 fall under valid class.
- **The answer is 'A'**

Example 3 [2]

- In an Examination, a candidate has to score a minimum of 24 marks in order to clear the exam. The maximum that he can score is 40 marks. Identify Valid Equivalence values if the student clears the exam
- a) 22,23,26
- b) 21,39,40
- c) 29,30,31
- d) 0,15,22

Solution

- The classes will be as follows:
Class I: values < 24 => invalid class
Class II: 24 to 40 => valid class
Class III: values > 40 => invalid class
- We need to identify Valid Equivalence values. Valid Equivalence values will be there in a Valid Equivalence class. All the values should be in Class II.
- **The answer is 'C'**

Example 4 [2]

- The Switch is switched off once the temperature falls below 18 and then it is turned on when the temperature is more than 21. When the temperature is more than 21. Identify the Equivalence values which belong to the same class.
- a) 12,16,22
- b) 24,27,17
- c) 22,23,24
- d) 14,15,19

Solution:

- We have to choose values from the same class (it can be a valid or invalid class). The classes will be as follows:
- Class I: less than 18 (switch turned off)
Class II: 18 to 21
Class III: above 21 (switch turned on)
- Only in Option “c”, all the values are from one class. Hence the **answer is ‘C’**. (Please note that this question does not talk about valid or invalid classes. It is only about values in the same class)

Example 5 [2]

In a system designed to work out the taxes to be paid:

- An employee has £4000 of salary tax-free.
- The next £1500 is taxed at 10%.
- The next £28000 after that is taxed at 22%.
- Any further amount is taxed at 40%.

To the nearest whole pound, which of these groups of numbers fall into three DIFFERENT equivalence classes?

- a) £4000; £5000; £5500
- b) £32001; £34000; £36500
- c) £28000; £28001; £32001
- d) £4000; £4200; £5600

Solution:

- The classes will be as follows:

Class I : 0 to £4000 => no tax

Class II : £4001 to £5500 => 10 % tax

Class III : £5501 to £33500 => 22 % tax

Class IV : £33501 and above => 40 % tax

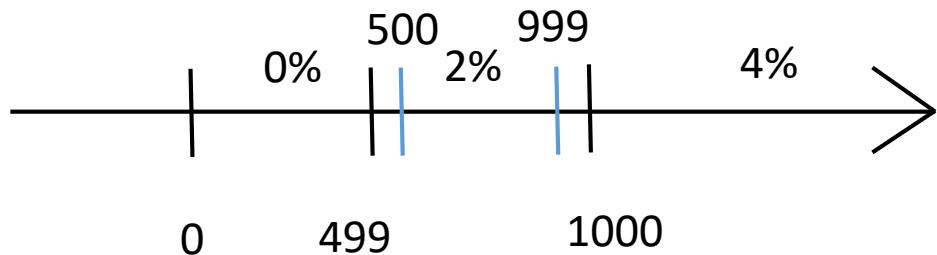
- Select the values that fall into three different equivalence classes.
Option 'd' has values from three different equivalence classes.
- **The answer is 'D'.**

Boundary value analysis (BVA)

Example 6 [5]

- Bank fee is 0% for balance less than 500 US\$, 2% for less than 1000 US\$, and 4% for 1000 US\$ or more. Which test inputs in US\$ would be selected using BVA considering valid boundary values?
 - a) 0.00, 0.01, 499.99, 500.00, 500.01, 999.99, 1000.00, 1000.01
 - b) 0.00, 499.99, 500.00, 999.99, 1000.00
 - c) -0.01, 499.99, 500.00, 999.99, 1000.00
 - d) 0.00, 500.00, 500.01, 1000.00, 1000.01

Bank fee is 0% for balance less than 500 US\$, 2% for less than 1000 US\$, and 4% for 1000 US\$ or more. Which test inputs in US\$ would be selected using BVA considering **valid** boundary values?



- a) ~~0.00, 0.01, 499.99, 500.00, 500.01, 999.99, 1000.00, 1000.01~~
- b) **0.00, 499.99, 500.00, 999.99, 1000.00**
- c) ~~0.01, 499.99, 500.00, 999.99, 1000.00~~
- d) ~~0.00, 500.00, 500.01, 1000.00, 1000.01~~

Example 7 [2]

- A program validates numeric fields as follows: values less than 10 are rejected, values between 10 and 21 are accepted, values greater than or equal to 22 are rejected. Which of the following covers the MOST boundary values?
- A) 9,10,11,22
- B) 9,10,21,22
- C) 10,11,21,22
- D) 10,11,20,21

Solution:

- Class I: values ≤ 9 \Rightarrow invalid class
Class II: 10 to 21 \Rightarrow valid class
Class III: values ≥ 22 \Rightarrow invalid class
- The boundaries can be identified as 9, 10, 21, and 22. These four values are in option 'b'.

Example 8 [2]

- In a system designed to work out the taxes to be paid:
- An employee has £4000 of salary tax-free.
- The next £1500 is taxed at 10%.
- The next £28000 after that is taxed at 22%.
- Any further amount is taxed at 40%.
- To the nearest whole pound, which of these is a valid Boundary Value Analysis test case?
 - a) £28000
 - b) £33501
 - c) £32001
 - d) £1500

Solution

- The classes will be as follows:

Class I : 0 to £4000 => no tax

Class II : £4001 to £5500 => 10 % tax

Class III : £5501 to £33500 => 22 % tax

Class IV : £33501 and above => 40 % tax

- We have to select a value which is a boundary value (start/end value). 33501 is a boundary value.
- **The answer is 'B'.**

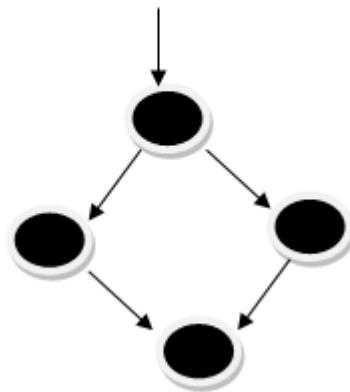
Literature

- [1] https://www.youtube.com/watch?v=sCA8xOg_S4
- [2] <https://www.softwaretestinghelp.com/istqb-exam-questions-equivalence-partitioning-boundary-value-analysis/>
- [3] <https://www.youtube.com/watch?v=peDxoB8Uy3M>

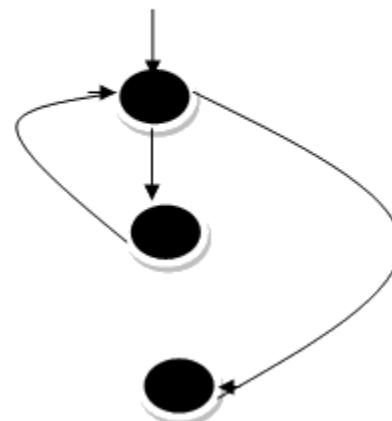
Testing – more examples

Conditions, Loops -> Graphs [1]

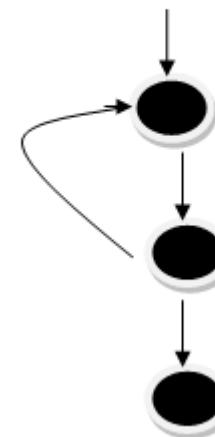
If-then-else:



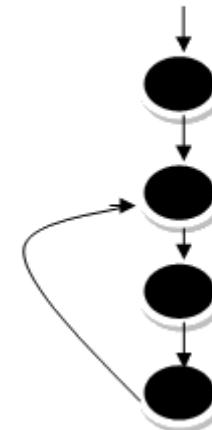
While:



Do-While:



For:



Example 1 [2]

- The employee goes to the shop and has to buy two items. If the total amount is more than \$100, the employee must pay himself. In the other case, if the total amount is less than \$100, the boss must pay for it. If the employee does not pay enough, the process is stopped.

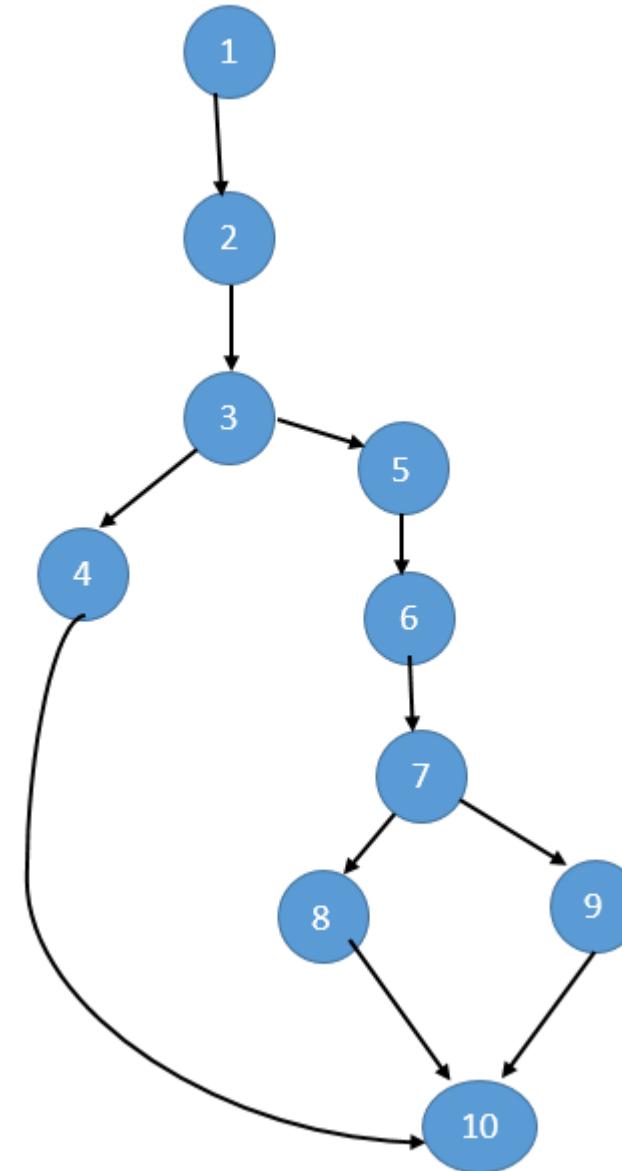
```
%initialization
prompt_1='Enter A: ';
prompt_2='Enter B: ';
A=input(prompt_1);
B=input(prompt_2);

%
C=A+B;

%
if C<100
    sprintf('It is Done')

else
    sprintf('Please, pay for items.')
    prompt_3='Enter:'
    D=input(prompt_3);
    if D>=C
        sprintf('It is pending')
    else
        sprintf('It is not enough. Sorry, try next time.')
    end
end
```

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths. Using the basic paths, please define test cases that allow us to traverse each of the basic paths.

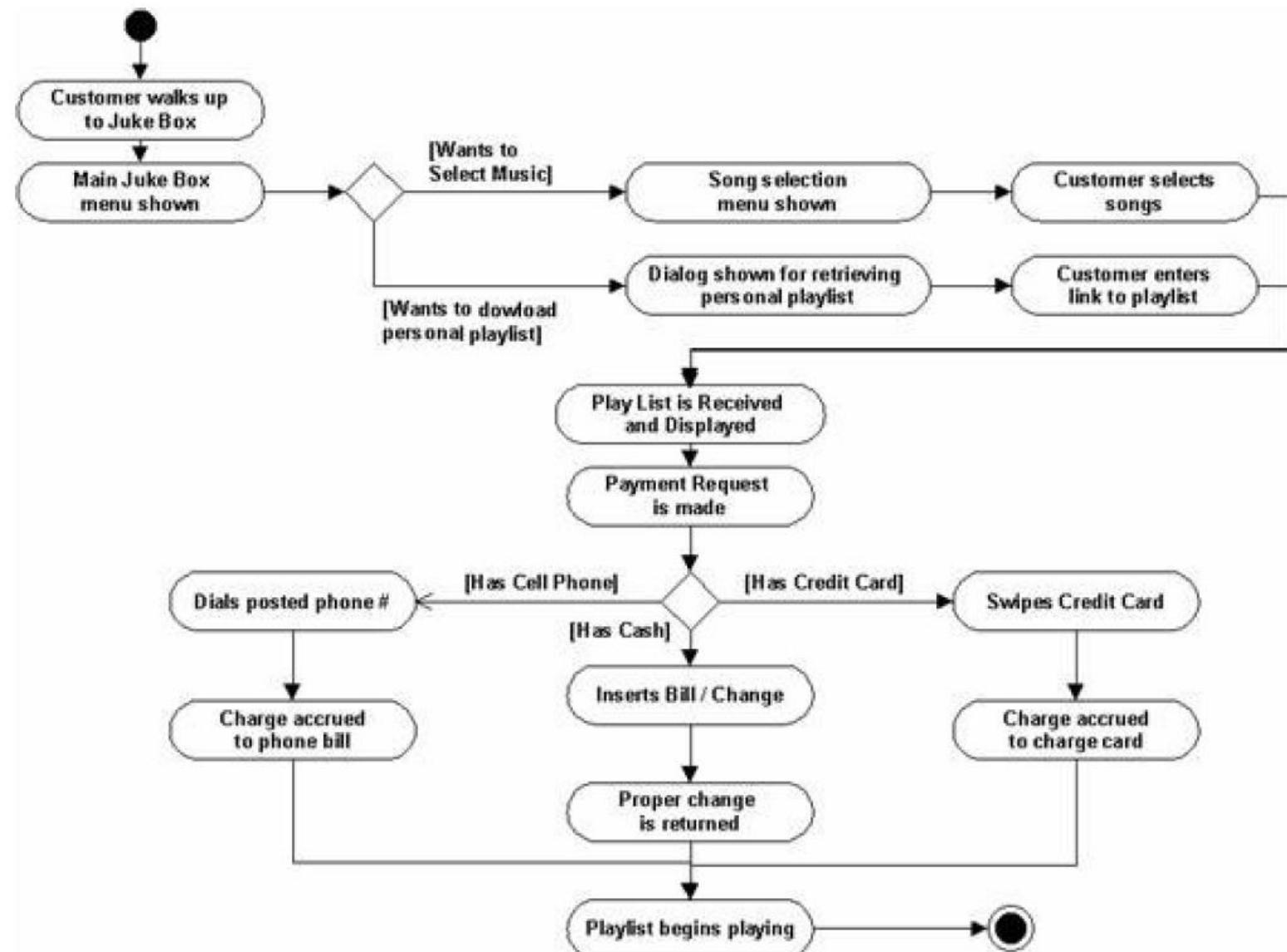


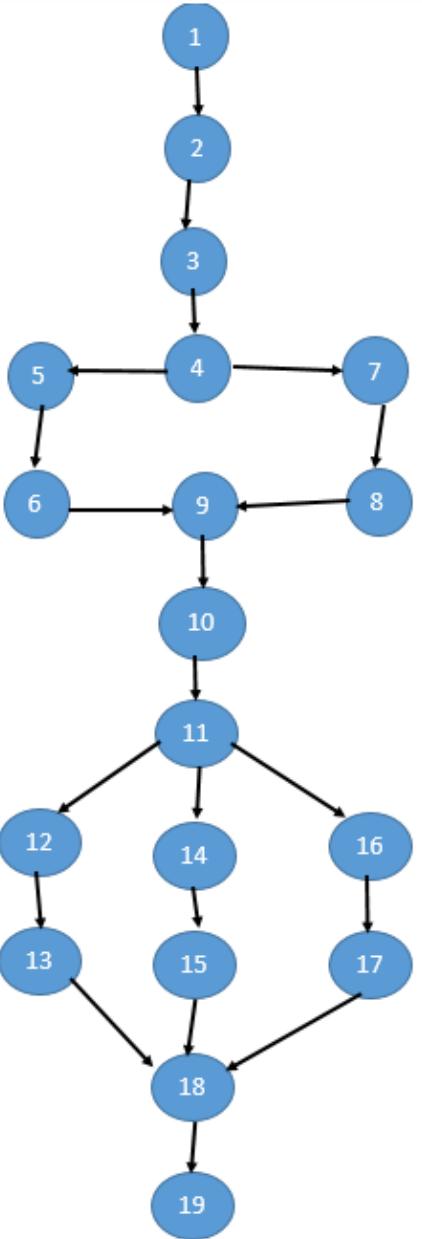
- $V(G) = 2 + 1 = 3$
- Basic paths:
- Path 1: 1-2-3-4-10
- Path 2: 1-2-3-5-6-7-8-10
- Path 3: 1-2-3-5-6-7-9-10

- Test Cases:
- Path 1:
 - Input parameters: A=30; B=50;
 - Output parameters: 'It is Done.'
- Path 2:
 - Input parameters: A=50; B=60; D=110
 - Output parameters: 'Please, pay for items.'; 'It is pending.'
- Path 3:
 - Input parameters: A=50; B=60; D=100;
 - Output parameters: 'Please, pay for items.' ; 'It is not enough. Sorry, try next time.'

Example 2

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths.



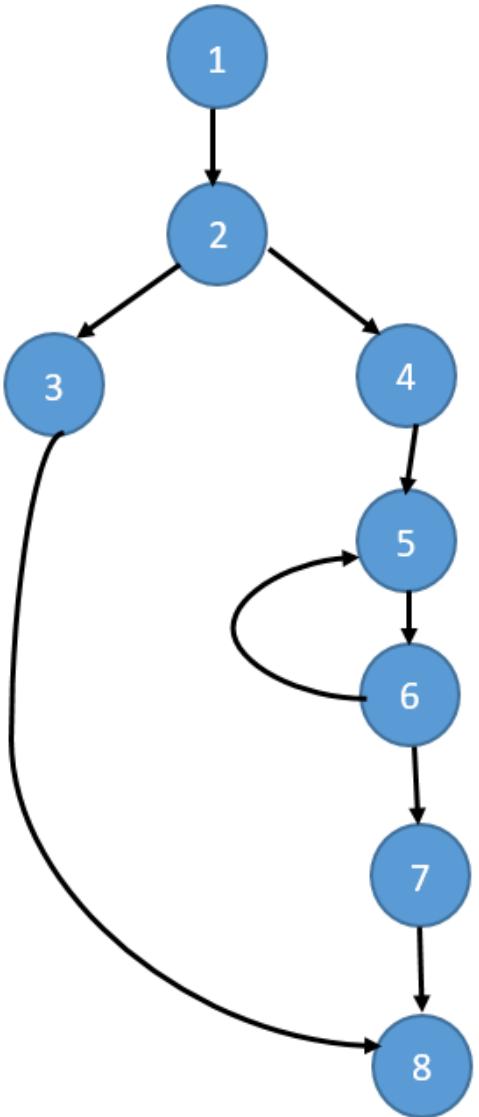


- $V(G) = 3+1 = 4$
- Basic paths:
- Path 1: 1-2-3-4-5-6-9-10-11-12-13-18-19
- Path 2: 1-2-3-4-7-8-9-10-11-12-13-18-19
- Path 3: 1-2-3-4-5-6-9-10-11-14-15-18-19
- Path 4: 1-2-3-4-5-6-9-10-11-16-17-18-19

Example 3

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths. Using the basic paths, please define test cases that allow us to traverse each of the basic paths.

```
%  
prompt_1='Enter n:'  
n=input(prompt_1);  
i=0;  
fact=1;  
%  
if n<1  
    sprintf('1')  
else  
    %  
    for i=1:n  
        %  
        fact=fact*i  
    end  
    sprintf('%d',fact)  
end
```



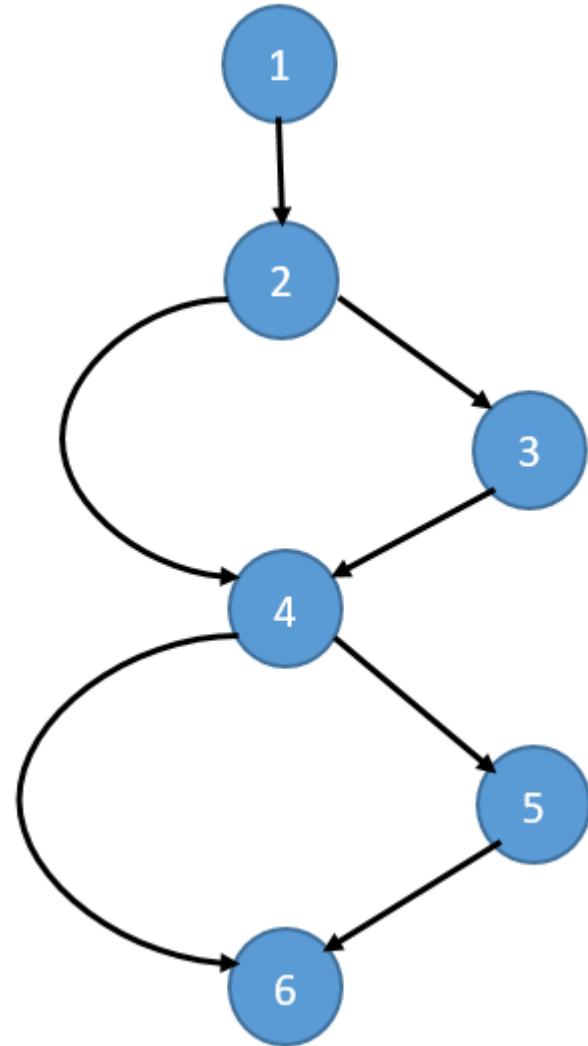
- $V(G) = 2 + 1 = 3$
- Basic paths:
- Path 1: 1-2-3-8
- Path 2: 1-2-4-5-6-7-8
- Path 3: 1-2-4-5-6-5-6-7-8

- Test cases:
- Path 1:
 - Input parameters: n=0
 - Output parameters: '1'
- Path 2:
 - Input parameters: n=1
 - Output parameters: fact=1, '120'
- Path 3:
 - Input parameters: n=5
 - Output parameters: fact=1, fact=2, fact=6, fact=24, fact=120, '120'

Example 4

- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths. Using the basic paths, please define test cases that allow us to traverse each of the basic paths.

```
%  
prompt_1='Enter X:';  
prompt_2='Enter Y:';  
X=input(prompt_1);  
Y=input(prompt_2);  
%  
if X+Y>100  
    sprintf('Greater')  
end  
if X>50  
    sprintf('X > 50')  
end
```



- $V(G) = 2 + 1 = 3$
- Basic paths:
- Path 1: 1-2-4-6
- Path 2: 1-2-3-4-6
- Path 3: 1-2-4-5-6

- Test cases:
- Path 1:
 - Input parameters: X=30; Y=20;
 - Output parameters:
- Path 2:
 - Input parameters: X=40; Y=70;
 - Output parameters: 'Greater'
- Path 3:
 - Input parameters: X=55; Y=40
 - Output parameters: 'X>50'

Example 5 [3]

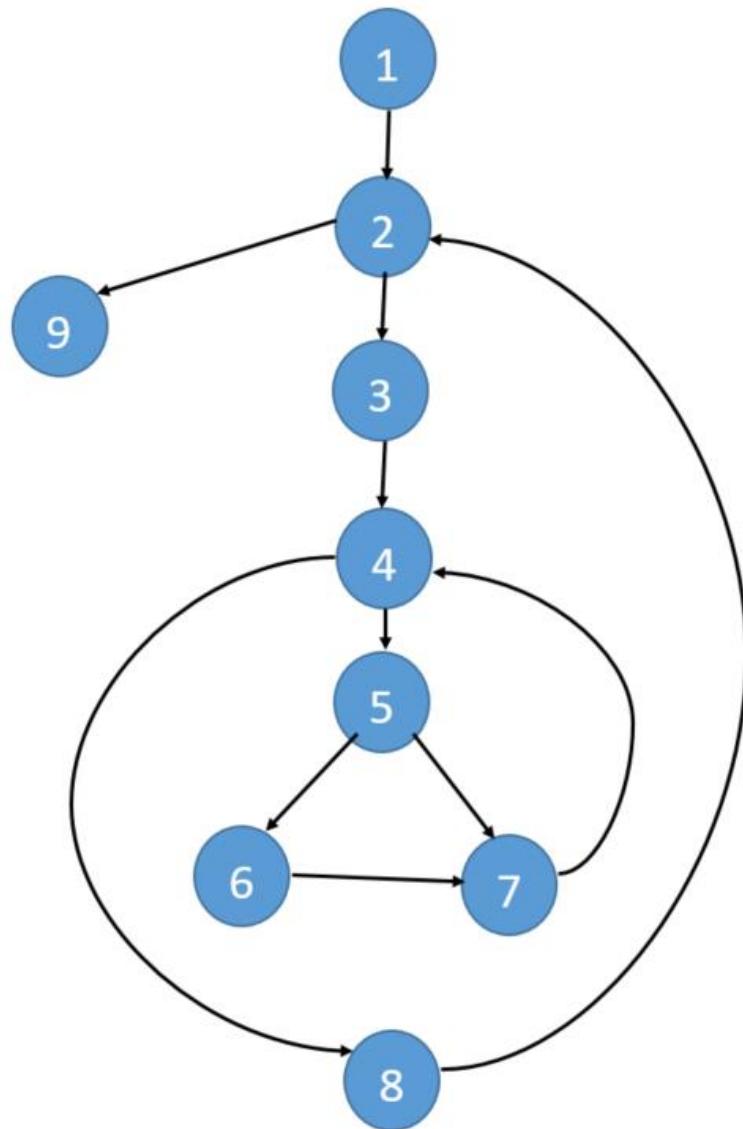
- Program to sort the elements of an array in descending order.
- Draw the corresponding graph of the control flow, determine the cyclomatic complexity, list all the basic (independent) paths.

```
%% Example 8
A=[2,5,1,8]
i=1;
n=numel(A);

while i<=n-1
    j=1+i;

    while j<=n
        if A(i) < A(j)
            t=A(i);
            A(i)=A(j);
            A(j)=t;
        end
        j=j+1;
    end
    i=i+1;
end

A
```



- $V(G) = 3 + 1 = 4$
- Basic paths:
- Path 1: 1-2-9
- Path 2: 1-2-3-4-8-2-9
- Path 3: 1-2-3-4-5-7-4-8-2-9
- Path 4: 1-2-3-4-5-6-7-4-8-2-9

Literature

- [1] <https://www.educba.com/cyclomatic-complexity/>
- [2] <https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/>
- [3] <https://medium.com/@danielsolano /white-box-test-cyclomatic-complexity-276fb5803a04>

Equivalence partitioning

- Equivalence partitioning is a procedure for determining the classes of input parameters from which to select test data.
- An equivalence class is a group of valid or invalid input conditions.
- Depending on the input conditions, we define the equivalent classes as follows:
 - The input condition is field-specified - specifies one valid and two invalid equivalent classes.
 - The input condition is a specific value - it specifies one valid and two invalid equivalent classes.
 - The input condition is determined by the membership of the set - it specifies one valid and one invalid equivalent class.
 - Boolean - Specifies one valid and one invalid equivalent class.

Boundary value analysis

- Boundary value analysis – (BVA) leads to the selection of test cases based on boundaries of input data.
 - Eg: the input condition is determined by the region bounded by a and b, - test cases should use the values a and b and the lower and higher values.
 - ...
- Testing with a BVA is not only about the input but also the output!

Example 1 [1] (Boundary Value Analysis)

- Suppose you have very important tool at office, accepts valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Define valid and invalid partitions. Write Test Cases for Valid partition value, Invalid partition value and exact boundary value. Define the test scenario.

- Valid range 8-12,
- Invalid range 7 or less than 7
- Invalid range 13 or more than 13

- Define registered users that will be used for the test scenario:
- Name: Pero_Peric1
 - Password: 2021_2
- Name: Pero_Peric2
 - Password: 2021_202
- Name: Pero_Peric3
 - Password: 2021_2022
- Name: Pero_Peric4
 - Password: 2021_2022_20
- Name: Pero_Peric5
 - Password: 2021_2022_2023

- Name: Pero_P
 - Password: 2021_2022
- Name: Pero_Per
 - Password: 2021_2022
- Name: Pero_Peric_P
 - Password: 2021_2022
- Name: Pero_Peric_Peric
 - Password: 2021_2022

- Test Cases 1: Consider password length less than 8.
 - Test scenario:
 - Password: 2021_2; Name: Pero_Peric1
- Test Cases 2: Consider password of length exactly 8.
 - Test scenario:
 - Password: 2021_202; Name: Pero_Peric2
- Test Cases 3: Consider password of length between 9 and 11.
 - Test scenario:
 - Password: 2021_2022; Name: Pero_Peric3
- Test Cases 4: Consider password of length exactly 12.
 - Test scenario:
 - Password: 2021_2022_20; Name: Pero_Peric4
- Test Cases 5: Consider password of length more than 12.
 - Test scenario:
 - Password: 2021_2022_2023; Name: Pero_Peric5

- Test Cases 6: Consider name length less than 8.
 - Test scenario:
 - Password: 2021_2022; Name: Pero_P
- Test Cases 7: Consider name of length exactly 8.
 - Test scenario:
 - Password: 2021_2022; Name: Pero_Per
- Test Cases 8: Consider name of length exactly 12.
 - Test scenario:
 - Password: 2021_2022; Name: Pero_Peric_P
- Test Cases 9: Consider name of length more than 12.
 - Test scenario:
 - Password: 2021_2022; Name: Pero_Peric_Peric

Example 2

- The vehicle power limitation module contributes to a more comfortable ride. The input of this module is the accelerator pedal position. The position is detected by a sensor and converted into a number that can have an integer value in the range from zero to 255. When the position is below 127 the module does not change the value, and in the range between 127 and 255 the module changes the value, such that it multiplies every change by 0.5. In this way the pedal is less sensitive to movement when pressed halfway down. Suggest testing method and test cases!

- Boundary value analysis is proposed because here we have strictly defined boundaries.
- Test case 1: exact boundary value 127
- Test case 2: less than 127, for example input: 125, output: 125
- Test case 3: higher value than 127, for example input: 170, output: 85

Example 3 [2]

- Please, use equivalence partitioning method in this example. Define valid and invalid classes.
- “Next Date” program.
Three Input Fields: Month, Day, Year
Two Buttons: “Next Date”, “Close”
Works only on dates after the Gregorian calendar was adopted on 10/15/1582
(the last year needs to be 3000).
- Note:
- In the Julian calendar, leap years are all divisible by 4. The 1582 calendar reform of the new Gregorian calendar adds the condition that years divisible by 100 are leap years only if they are divisible by 400.
- Thus, since the reform the leap years in the Gregorian calendar are 1600, 2000, and 2400, and not the years 1700, 1800, 1900, and 2100 because they are not divisible by 400.

- Equivalence Classes
- **Month**
- Valid class 1 : 30 day months
- Valid class 2 : 31 day months
- Valid class 3 : February
- Invalid class 1 : $>= 13$
- Invalid class 2 : $<= 0$
- Invalid class 3 : Any non-integer
- Invalid class 4 : Empty
- Invalid class 5 : $>= 3$ integers

- **Day**
- Valid class 4: 1-28 (all months have these days)
- Valid class 5: 29 (some months have this day, some don't)
- Valid class 6: 30 (some months have this day, some don't)
- Valid class 7: 31 (some months have this day, some don't)
- Invalid class 6: $>= 32$
- Invalid class 7: $<= 0$
- Invalid class 8: Any non-integer
- Invalid class 9: Empty
- Invalid class 10: $>= 3$ integers

- Year
 - Valid class 8: Non-leap year
 - Valid class 9: Leap year
 - Valid class 10: Century non-leap year
 - Valid class 11: Century leap year
- Invalid class 11: ≤ 1581
- Invalid class 12: ≥ 3001
- Invalid class 13: Any non-integer
- Invalid class 14: Empty
- Invalid class 15: ≥ 5 integers

Literature

- [1] <https://slideplayer.com/slide/14393576/>
- [2] https://students.cs.byu.edu/~cs240ta/fall2018/rodham_files/25-black-box-testing/NextDateExample.pdf