Java SE 7

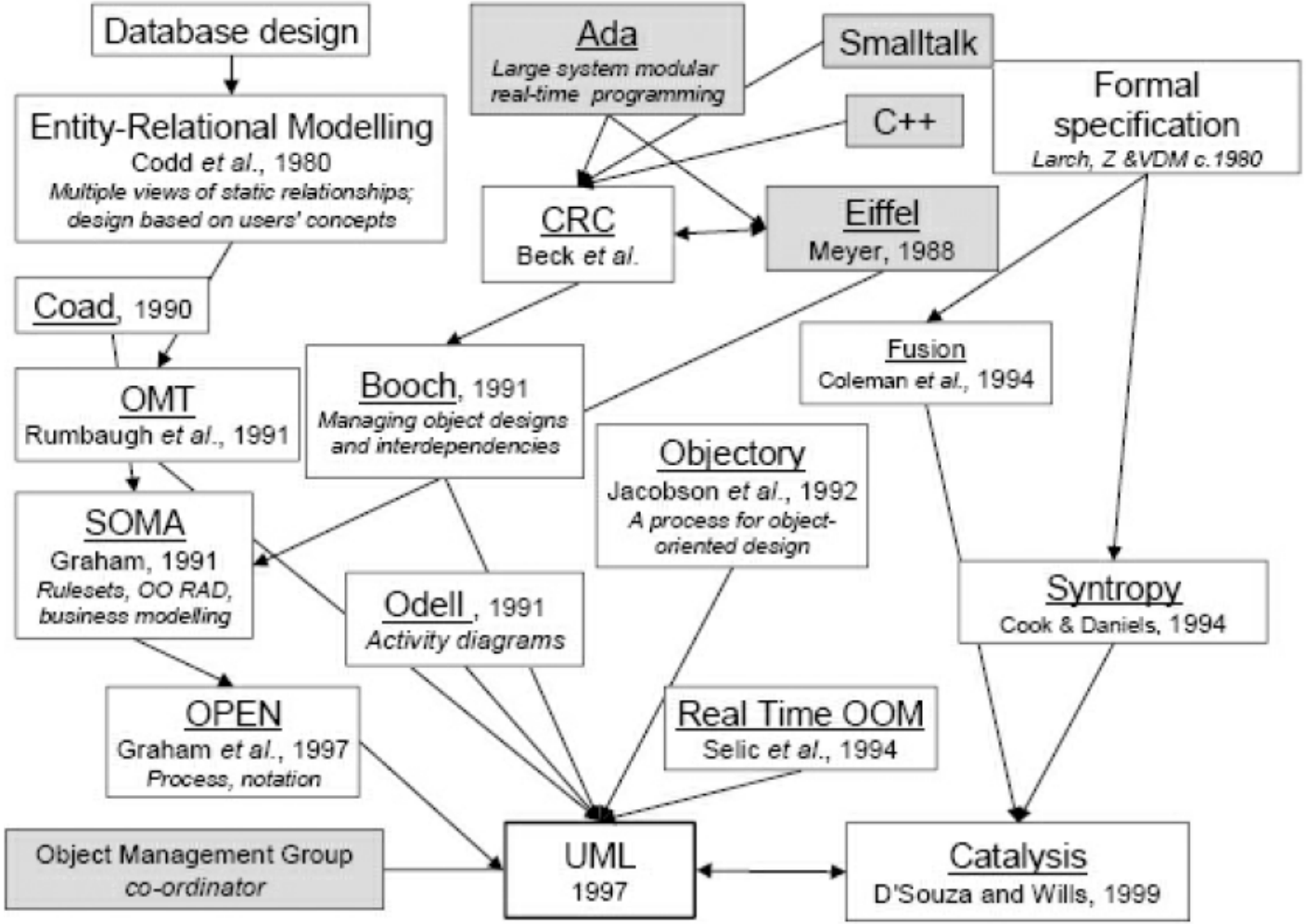
Module 5
UML

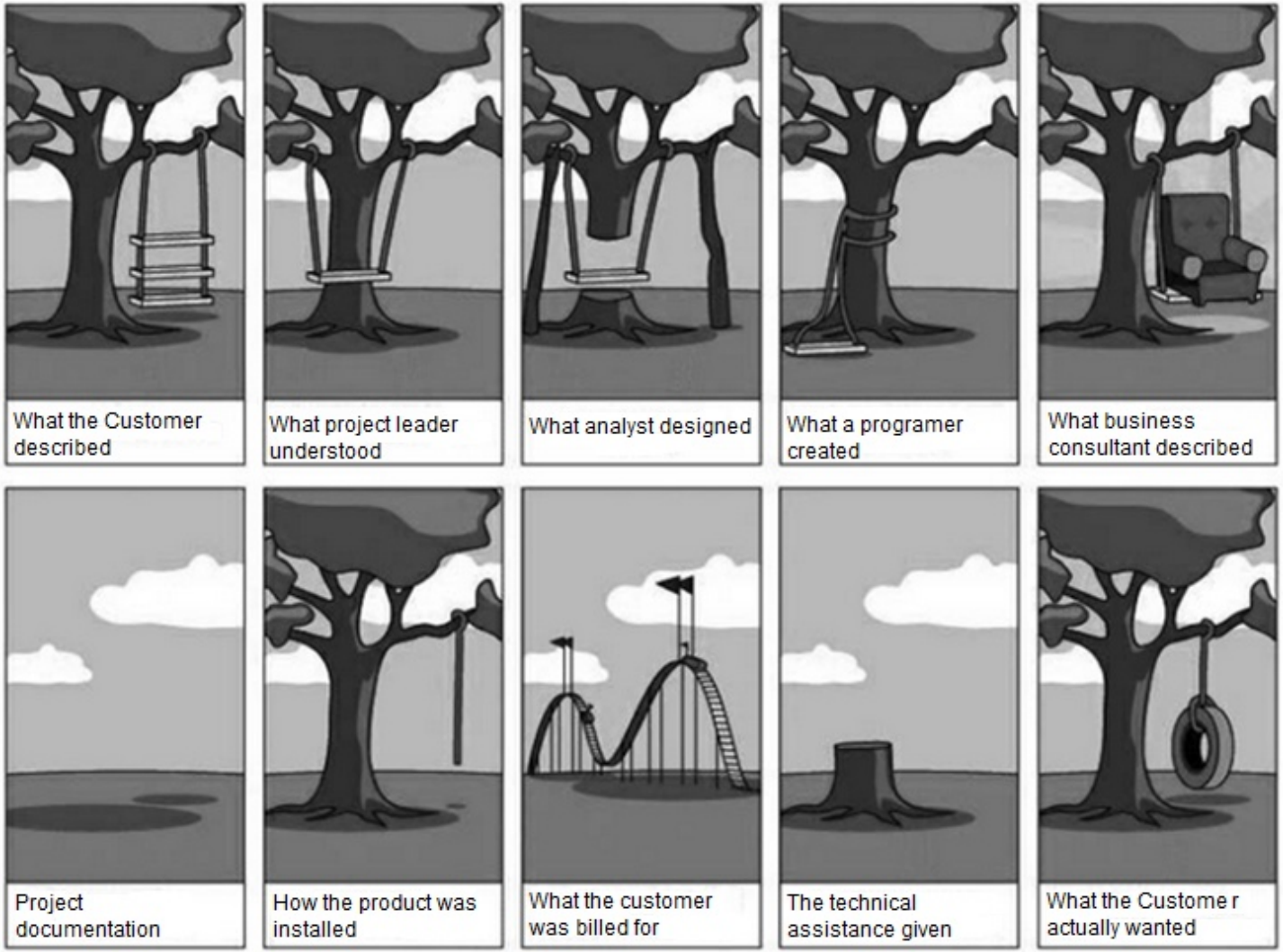
LUXOFT TRAINING

LXFT LISTED NYSE

Introduction to UML

1. UML is a unified modeling language.
2. This is a standard in OOP.
3. UML belongs to OMG.
4. UML profiles.
5. The “unified” language received some features of notations by Grady Booch, Jim Rumbaugh, Ivar Jacobson and many other people.


Database design
Ada
Large system modular real-time programming
Smalltalk
Formal specification
Larch, Z &VDM c.1980
C++
Entity-Relational Modelling
Codd et al., 1980
Multiple views of static relationships; design based on users' concepts
CRC
Beck et al.
Eiffel
Meyer, 1988
Coad, 1990
Booch, 1991
Managing object designs and interdependencies
Fusion
Coleman et al., 1994
OMT
Rumbaugh et al., 1991
Objectory
Jacobson et al., 1992
A process for object-oriented design
SOMA
Graham, 1991
Rulesets, OO RAD, business modelling
Odell, 1991
Activity diagrams
Syntropy
Cook & Daniels, 1994
OPEN
Graham et al., 1997
Process, notation
Real Time OOM
Selic et al., 1994
Object Management Group
co-ordinator
UML
1997
Catalysis
D'Souza and Wills, 1999


What the Customer described
What project leader understood
What analyst designed
What a programer created
What business consultant described
Project documentation
How the product was installed
What the customer was billed for
The technical assistance given
What the Custome r actually wanted

Design Problems

- Problems with communication and understanding caused by lack of clear specification for a product.

- Object-oriented programming languages are aimed at specification, visualization, design and documentation of all artifacts created during development.

Notaton

Notation (or *syntax* in other languages) stresses that UML is a graphical language and that models (diagrams) are not written, but drawn.

UML uses four types of notation elements:

- figures
- lines
- symbols
- legends

Software Tools

There are software tools that allow designing and “drawing” UML diagrams on PC (CASE tools):

- IBM Rational Rose
- Borland Together
- Gentleware Poseidon
- Microsoft Visio
- Telelogic TAU G2

Types of diagrams

There are **13 types** of diagrams in UML divided into three categories.

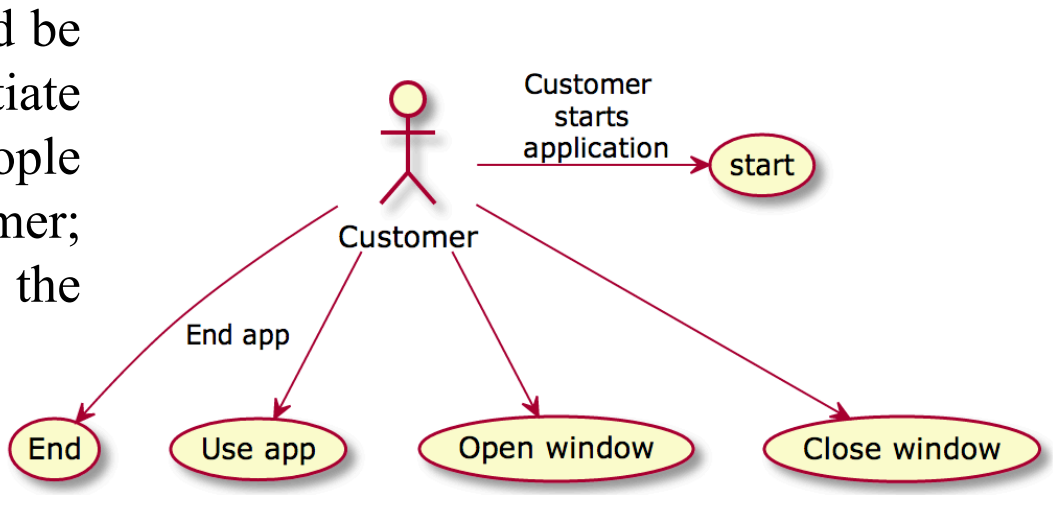
- **7** diagram types represent static structure of application (static diagrams).
- **3** diagram types represent behavioral information of system (behavioral diagrams).
- **3** diagram types represent physical aspects of system functioning (interaction diagrams).

Types of diagrams

- Class
- Object
- Use case
- Sequence
- State
- Deployment
- Activity

Use case diagram

In the diagram, a use case should be only persons (**Actors**) which initiate operations of the system: real people or actions performed on the timer; until such person does not "kick" the system, it will sleep.



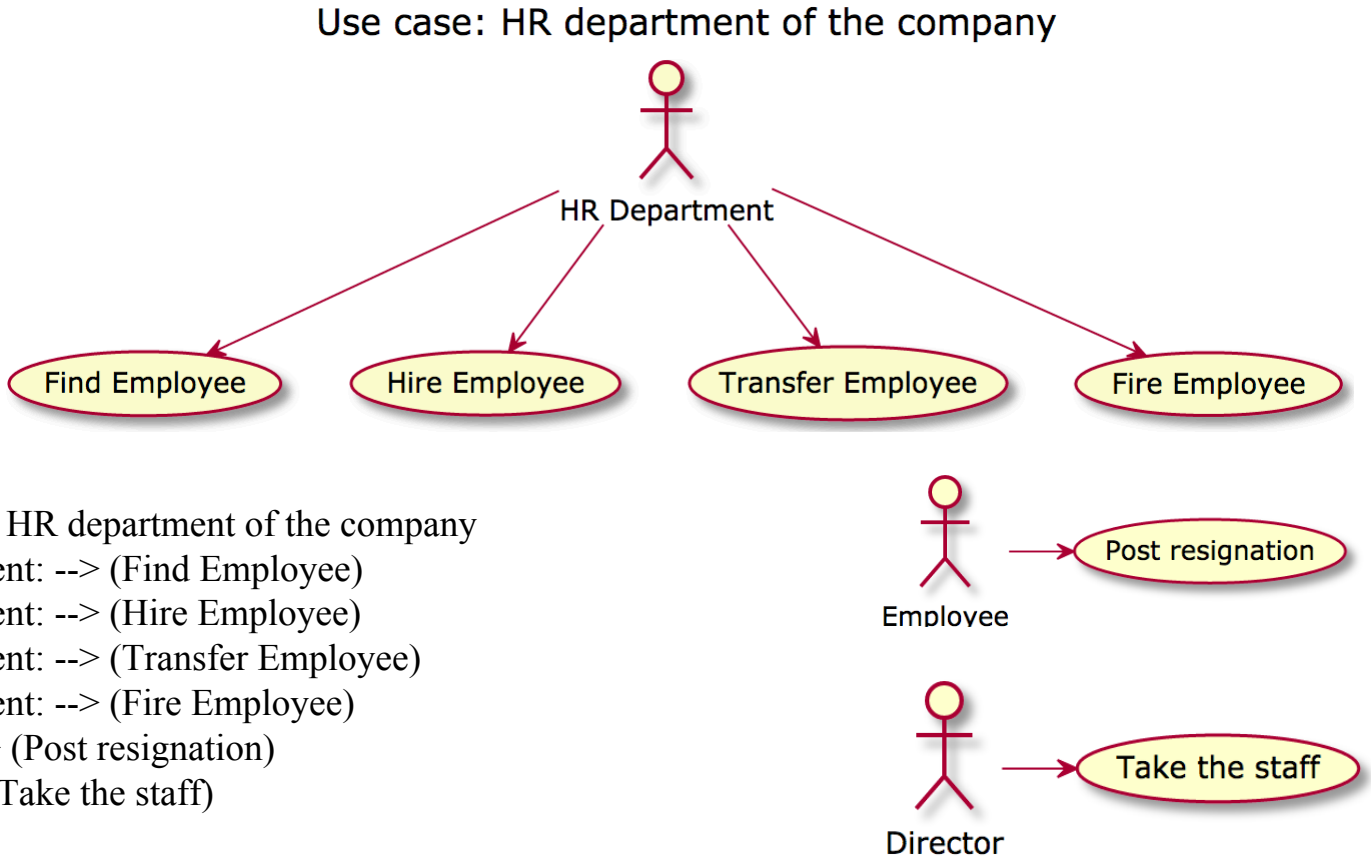

```
Customer -> (start) : Customer \n starts \n application
Customer --> (End) : End app
Customer --> (Use app)
Customer --> (Open window)
Customer --> (Close window)
```

Task Example: HR Department

- Create the project accounting system for a human resources company.

- The system must maintain a list of candidates, to take into account the process of interviewing and hiring.

Use case diagramm for HR Department



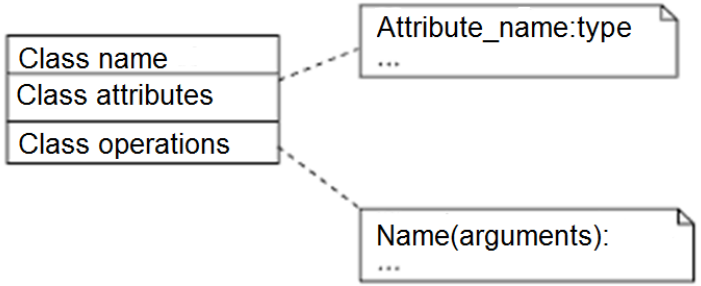
title Use case: HR department of the company
:HR Department: --> (Find Employee)
:HR Department: --> (Hire Employee)
:HR Department: --> (Transfer Employee)
:HR Department: --> (Fire Employee)
:Employee: -> (Post resignation)
:Director: -> (Take the staff)

Class Diagram

- Classes are building blocks of any object-oriented system.
- Information from class diagram is directly rendered in application source code.

Class Diagram

- On diagram, a class is drawn as a rectangle divided by horizontal lines into three parts.
- Class name is indicated in the first part.
- The second part contains the attributes of the class that characterize one or another class objects.
- The third part contains operations reflecting its behavior in the subject area model.

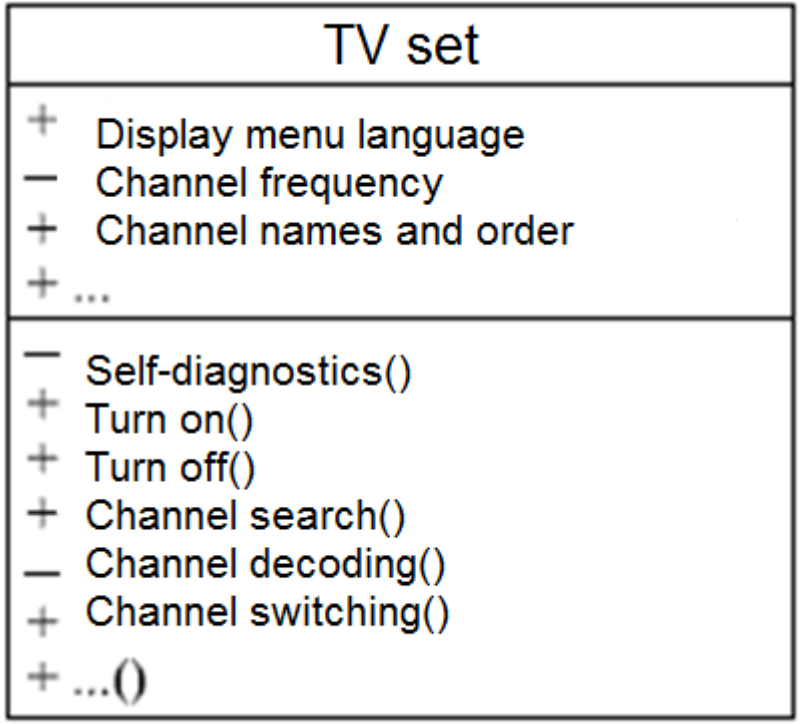


Class Diagram

- During the design, a class acts as a “black box”. The body of class methods is not specified in UML.
- Object encapsulation requires that all access modifiers should be specified.

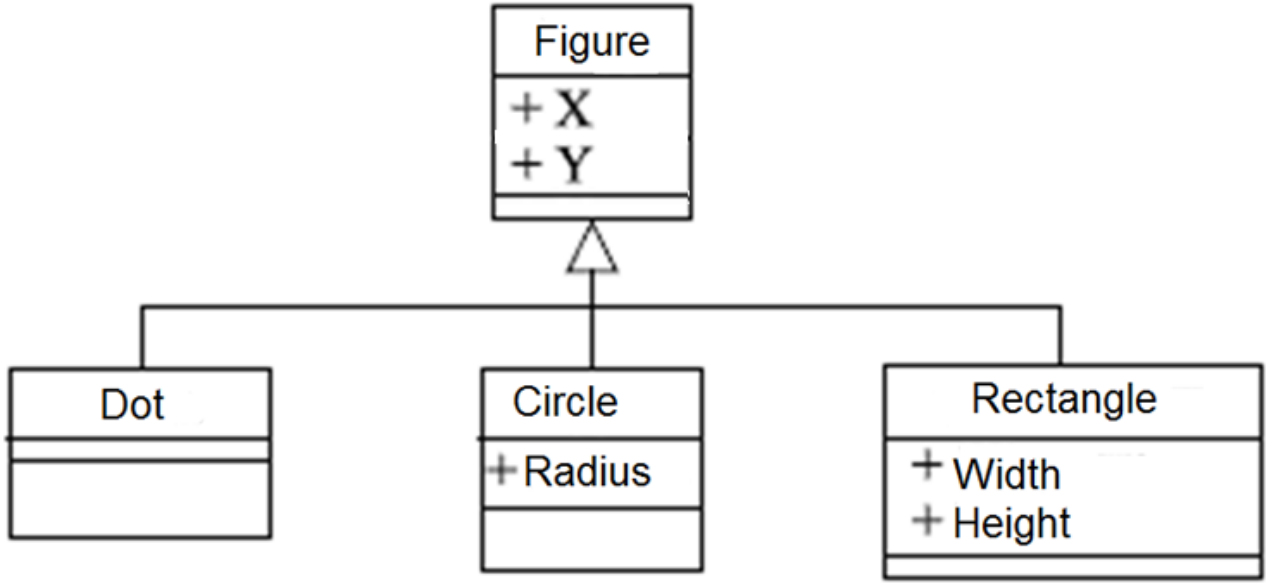
\+ **(public)** – public access.
- **(private)** – only operations of the same class.
**(protected)** – only operations of the same class and classes created on its basis.

Class Diagram



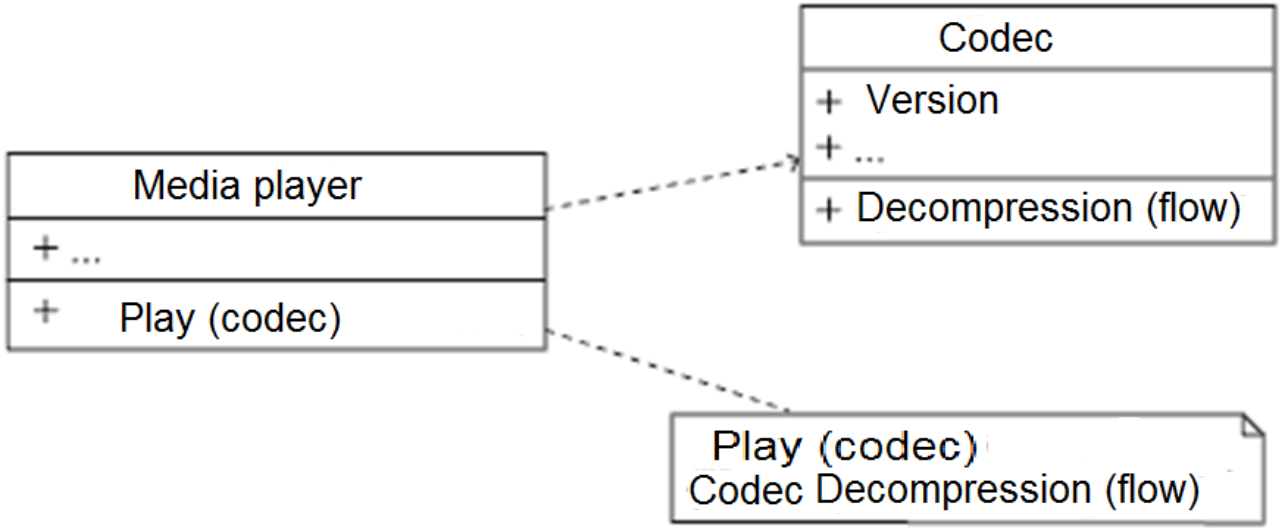
Class Diagram

- Inheritance is a relationship between a more general entity called the superclass and its specialized form called the subclass.



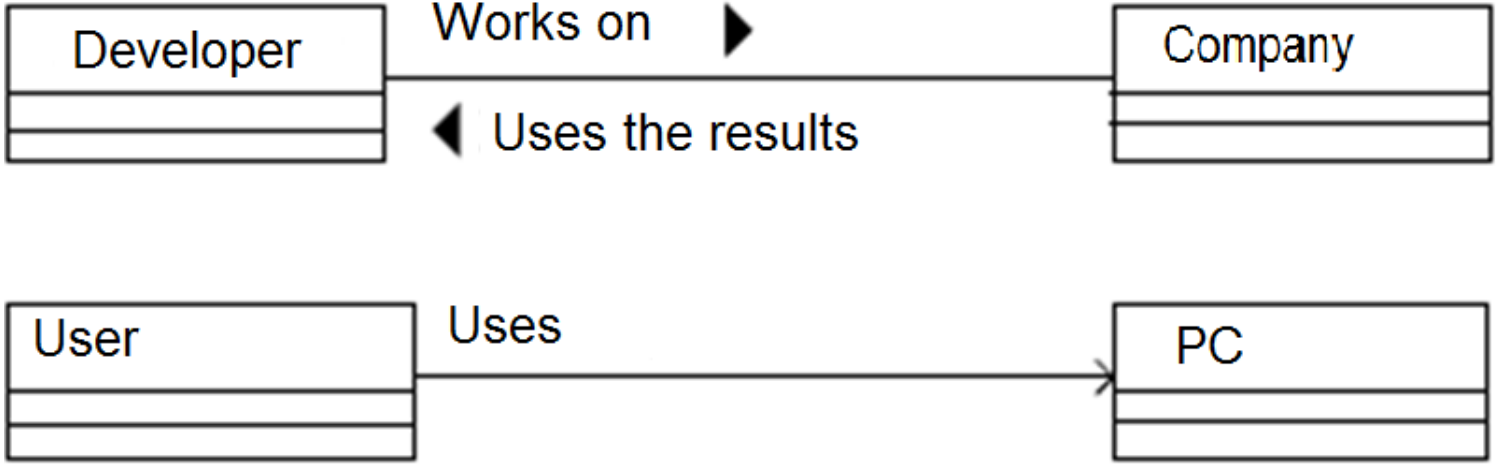
Class Diagram

- Dependency is a form of relationship between class objects, wherein class implementation of one object depends on specification for class operations of another object.



Class Diagram

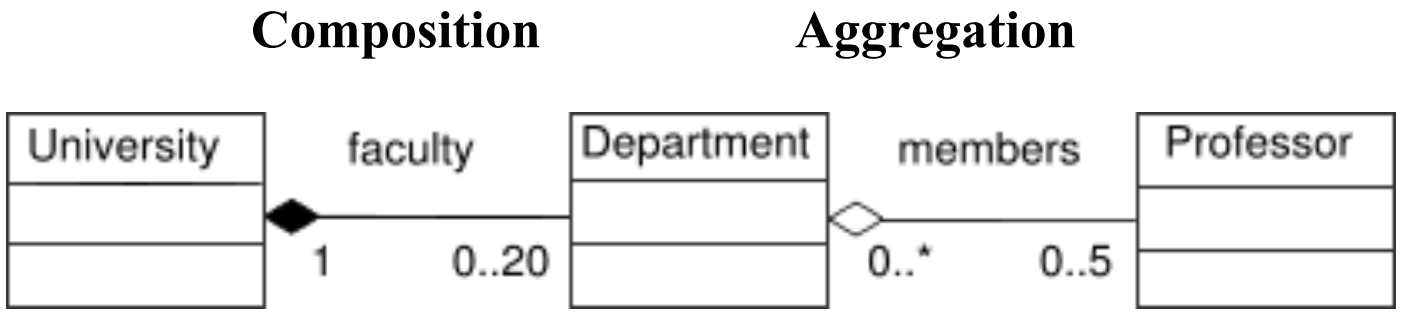
- Association is a link connecting objects.



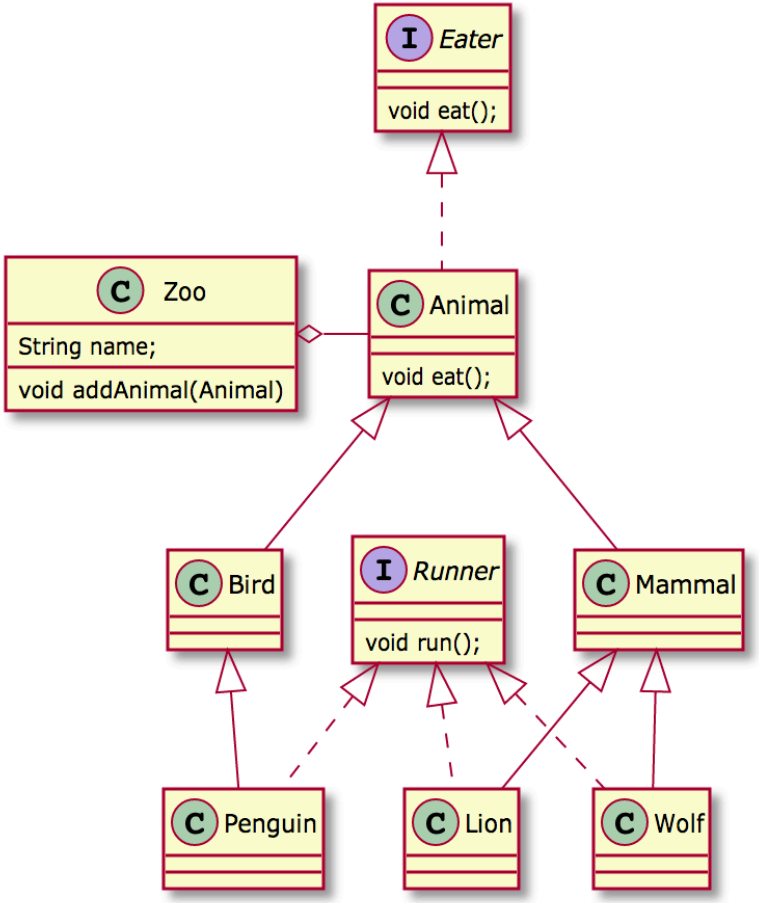
Class Diagram

Aggregation (aggregation via the link) - the ratio of "part-whole" between two peer entities, where one object (container) has a reference to another object. Both objects can exist independently: if the container is destroyed, its contents - no.

Composition - a more strict version of aggregation, when the object can only exist as part of the container. If the container is destroyed, then the included object is destroyed too.

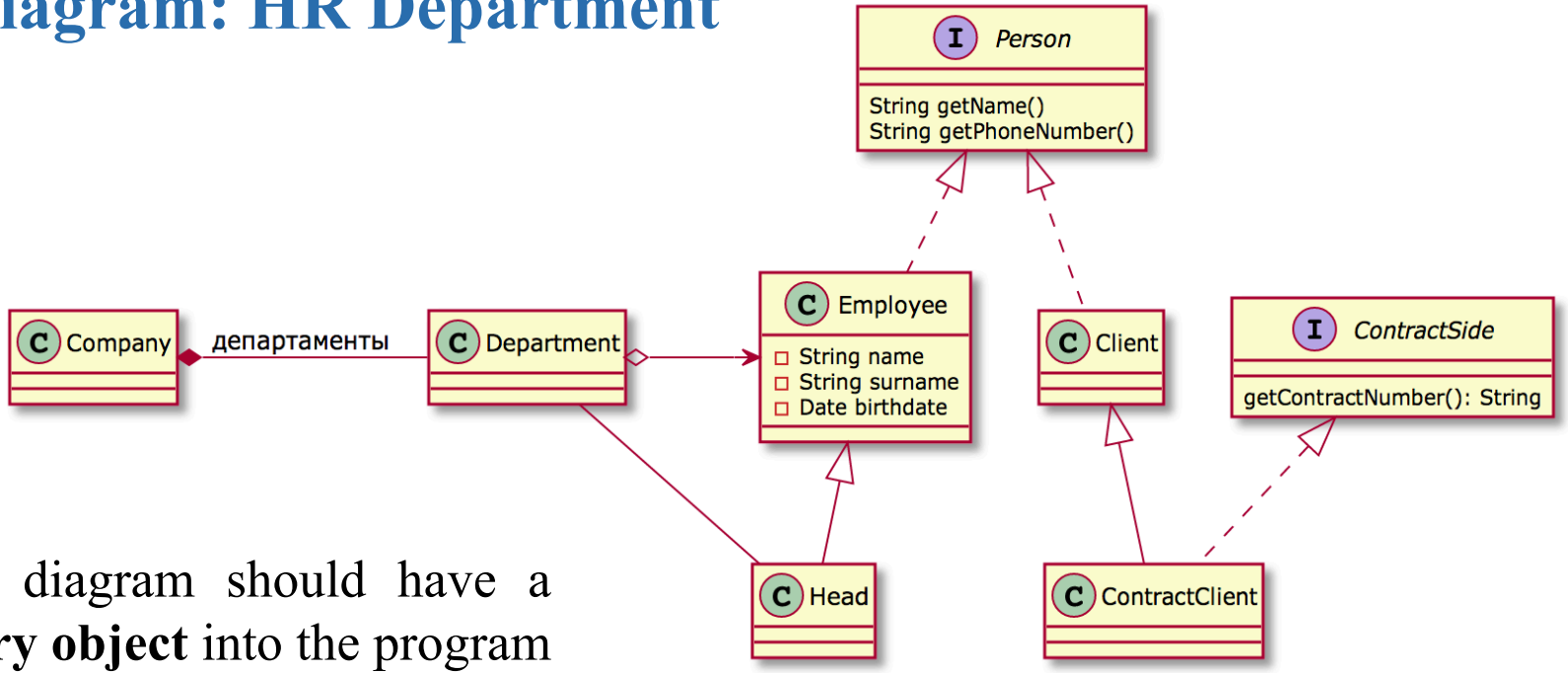


Class Diagram




```
interface Eater {
    void eat();
}
class Animal implements Eater {
    void eat();
}
Zoo o- Animal
class Zoo {
    String name;
    void addAnimal(Animal)
}
class Mammal extends Animal
class Lion extends Mammal
class Wolf extends Mammal
class Wolf implements Runner
class Lion implements Runner

interface Runner {
    void run();
}

class Bird extends Animal
class Penguin extends Bird
class Penguin implements Runner
```

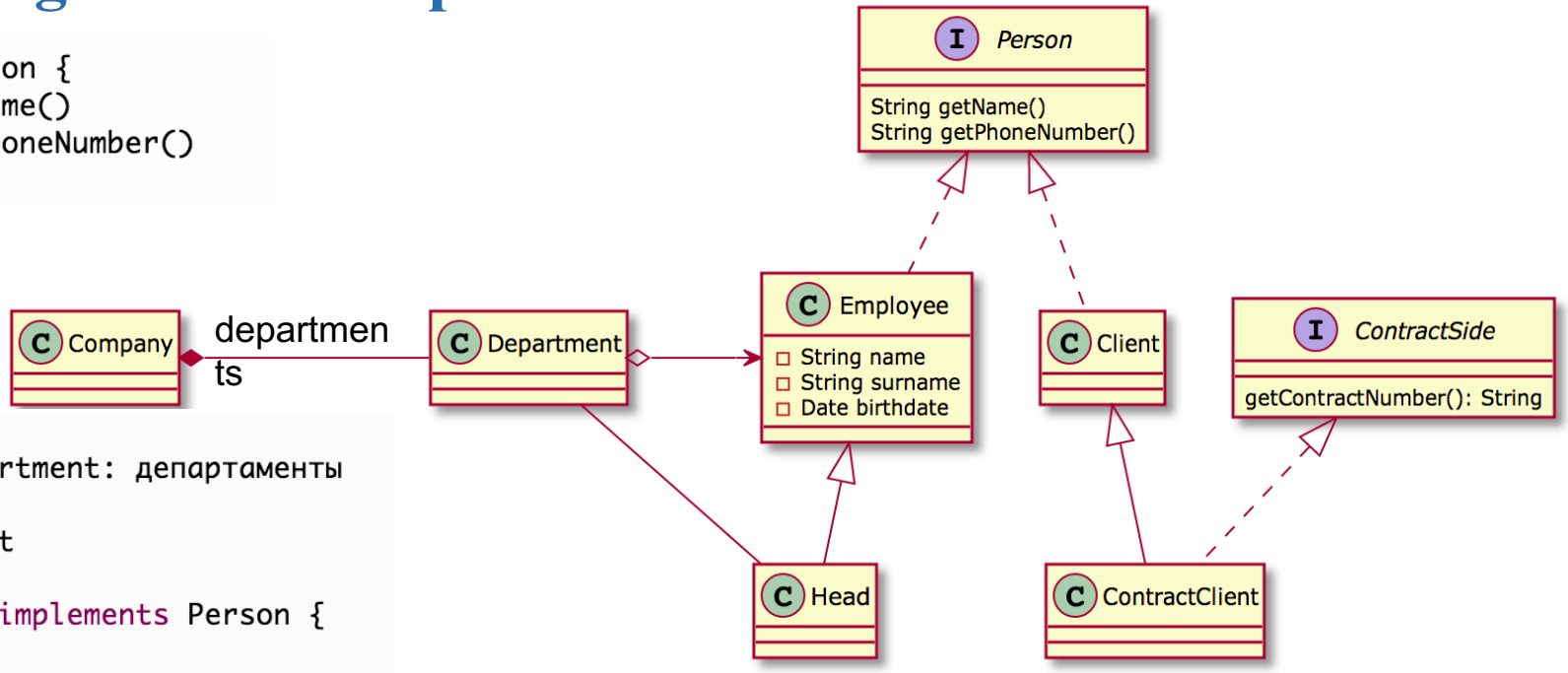
Class Diagram: HR Department



The class diagram should have a **single entry object** into the program (for example, the whole company, the whole restaurant, etc.)

Class Diagram: HR Department

```
interface Person {
  String getName()
  String getPhoneNumber()
}
```

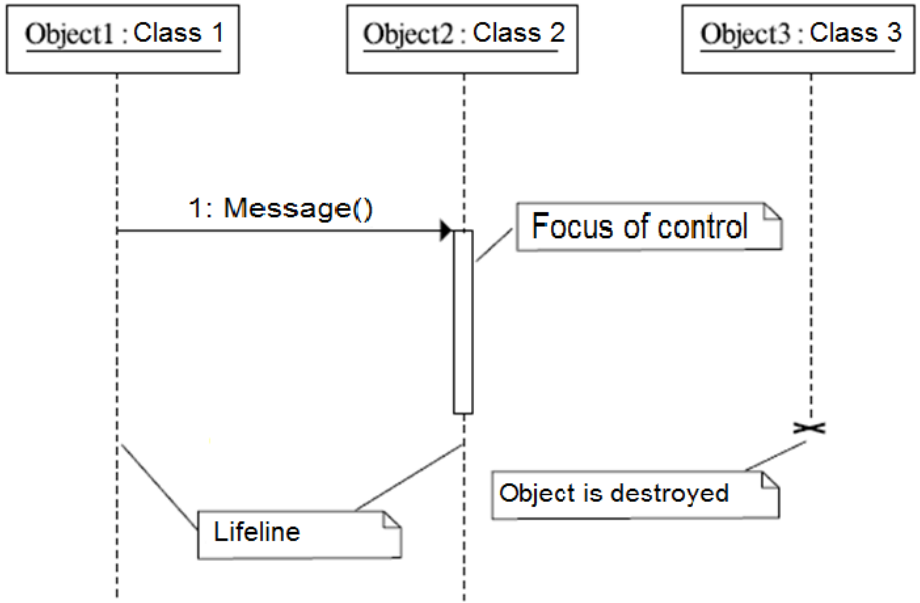



```
class Company
Company *- Department: департаменты

class Department

class Employee implements Person {
  -String name
  -String surname
  -Date birthdate
}

class Head extends Employee
Department -- Head
Department o-> Employee
```

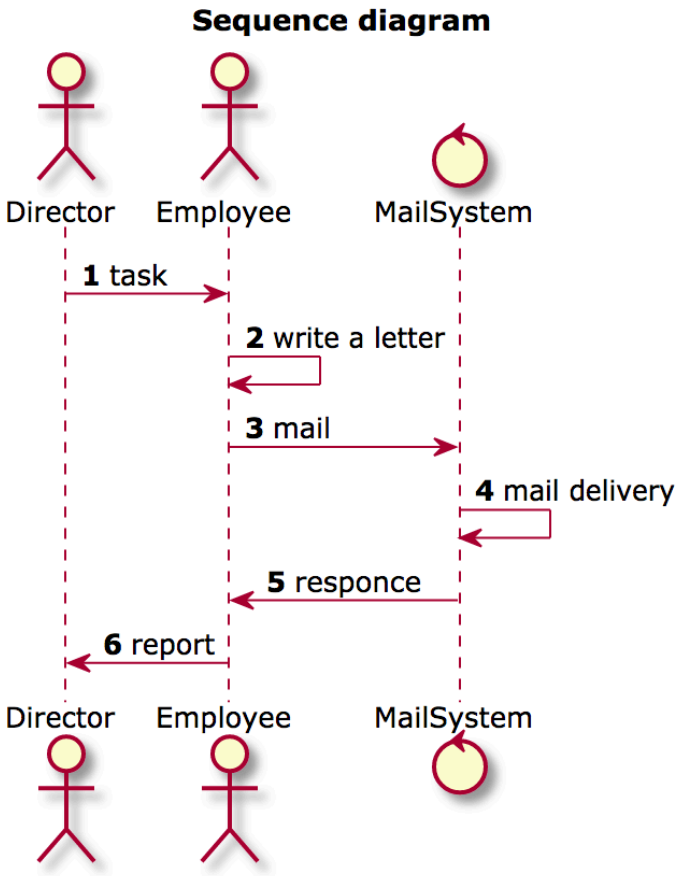
```
interface ContractSide {
  getContractNumber(): String
}

class Client implements Person
class ContractClient extends Client
class ContractClient implements ContractSide
```

Sequence Diagram

- Sequence diagram shows the sequence in which interacting objects exchange messages.
- Object lifeline is a line that represents the existence of an object during some time.



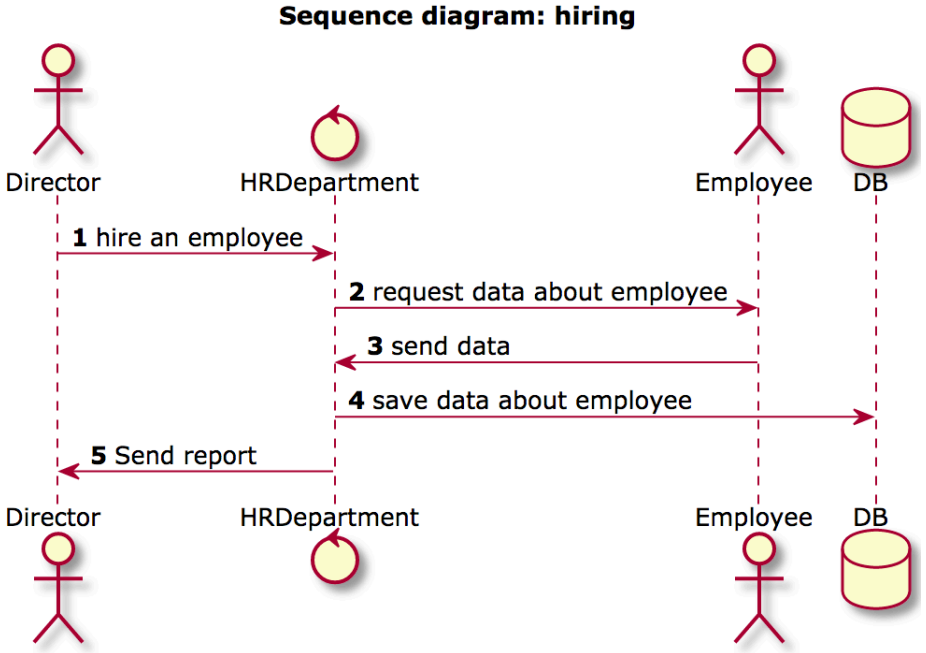
Sequence Diagram



Sequence diagram describes some process within the system, such as the process of selling goods or the process of work on a task (see example).

Sequence diagram shows how the users interact with the program components.

Sequence Diagram for HR hire process

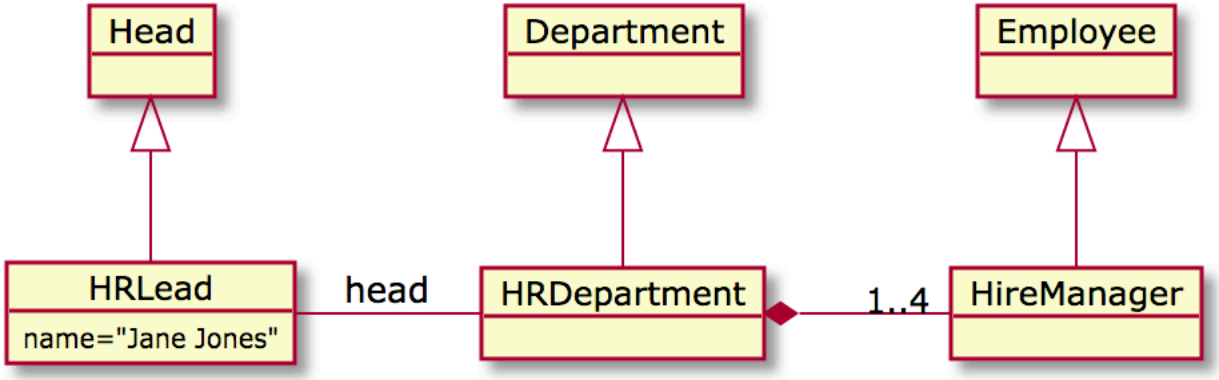



```
title Sequence diagram: hiring
autonumber
actor Director
control HRDepartment
actor Employee
database DB
```

```
Director -> HRDepartment: hire an employee
HRDepartment -> Employee: request data about employee
Employee -> HRDepartment: send data
HRDepartment -> DB: save data about employee
HRDepartment -> Director: Send report
```

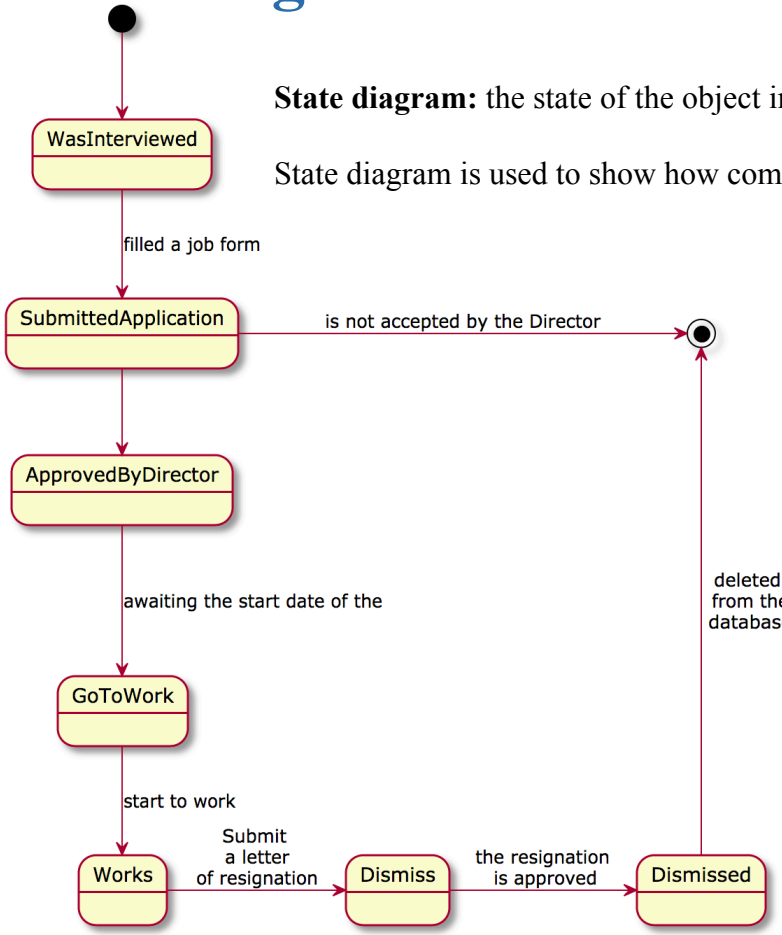
Object Diagram

Objects diagrams are representing the static view of the system. This static view is a snapshot of the system at a particular moment.



Object diagram: used for system initialization and testing.
Example: filling system with the test data, to drive the unit-tests.

State Diagram



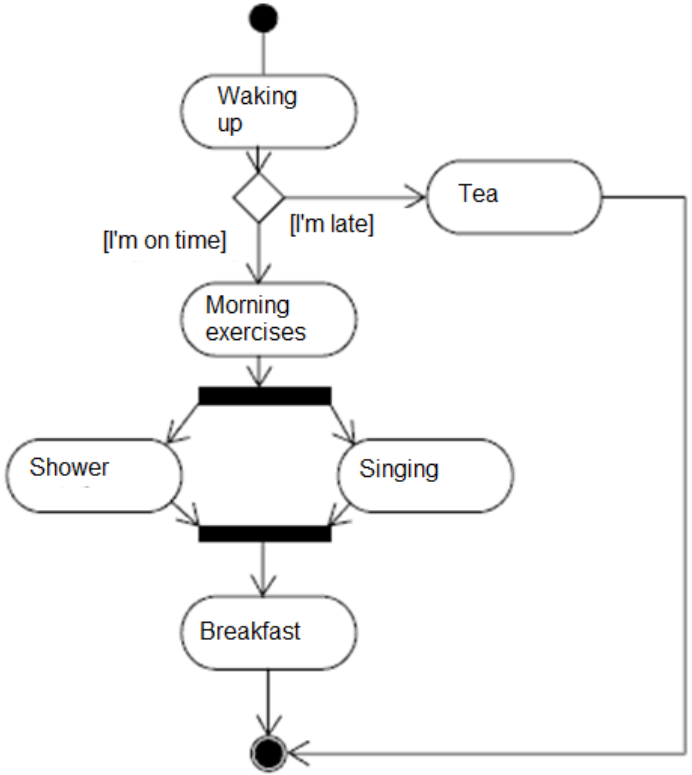
State diagram: the state of the object in the program.

State diagram is used to show how complex objects work in the program. It models the lifetime of an object.

```
title State diagram: Employee
[*] --> WasInterviewed
WasInterviewed --> SubmittedApplication: filled a job form
SubmittedApplication --> ApprovedByDirector
SubmittedApplication -> [*]: is not accepted by the Director
ApprovedByDirector --> GoToWork: awaiting the start date of the
GoToWork --> Works: start to work
Works -> Dismiss: Submit\n a letter\n of resignation
Dismiss -> Dismissed: the resignation\n is approved
Dismissed -u-> [*]: deleted\n from the\n database
```

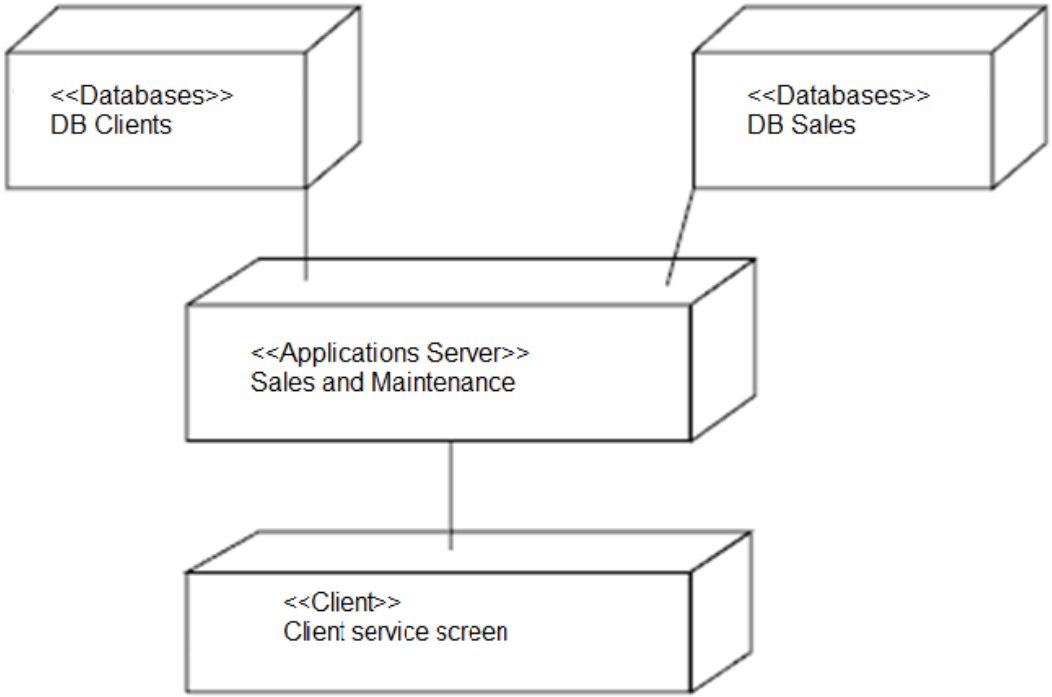
Activity Diagram

Activity diagrams are graphical representations of algorithms of classes operations.



Deployment Diagram

In case of complex IT infrastructure (databases, web services, etc.), it is very useful to have a graphical representation of this structure.



Lab guide

Tasks for UML design:

1) University: create a system for accounting students, attendance, academic achievements. Optional: it must contain reviews of teachers.

2) Shop: should keep information about the goods and sales of the goods. Optional: The system should provide discounts for regular customers.

3) Clinic: to create a system for the appointment to the doctors, to the time of the doctors. Optional: keeping patient cards by physicians.

4) Airport: you must manage the landing of the aircrafts, with their distribution in time and runways. Optional: notifications about flight delays to passengers, via SMS.

5) Restaurant: to create a system of accounting for the employment of tables, selection of dishes from the menu, bill payment. Optional: allow the reservation of tables.

Exercise: use jRunner to create UML-diagrams

Java School | **jRunner**: task execution environment. | Attendee: **Vladimir Sonkin**

Section:

- UML-example
- UML
- Java basics
- Main classes

Task:

- Use case: HR department
- Sequence diagram
- Class diagram
- State diagram: Employee
- Object diagram
- Activity diagram
- Typical errors in UML
- Tasks for UML design

Use case: HR department

☐ Task finished!

```
1  title Use case: HR department of the company
2  :HR Department: --> (Find Employee)
3  :HR Department: --> (Hire Employee)
4  :HR Department: --> (Transfer Employee)
5  :HR Department: --> (Fire Employee)
6  :Employee: -> (Post resignation)
7  :Director: -> (Take the staff)
8
9
```


LUXOFT TRAINING

PlantUML: tool to build UML diagrams (plantuml.com)