INTRODUCTION TO UML

INTRODUCTION TO UML

Based on the book UML distilled by Martin Fowler

INTRODUCTION TO UML

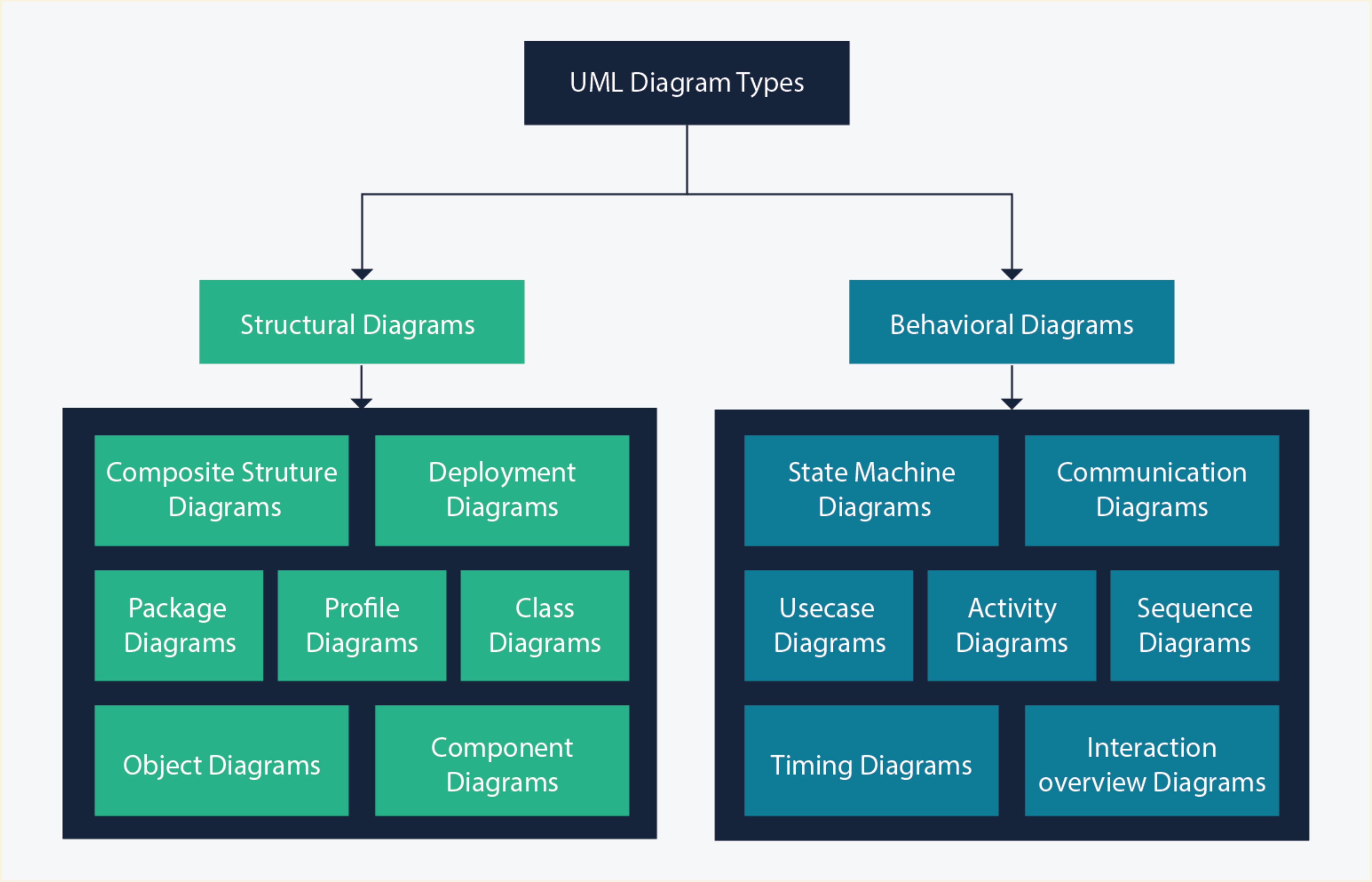
Based on the book UML distilled by Martin Fowler

- Overview
- Diagrams
- Exercises

WHAT IS UML ?

- Family of graphical notations
- Was built by Objet Management Group in 97
- Defines **notation** and a **meta-model**
- **2.0** defines **14** official diagrams
- UML is a mix of *prescriptive (formal syntax)* and *descriptive (shaped by usage and conventions)*
- Defines no mapping to existing programming languages

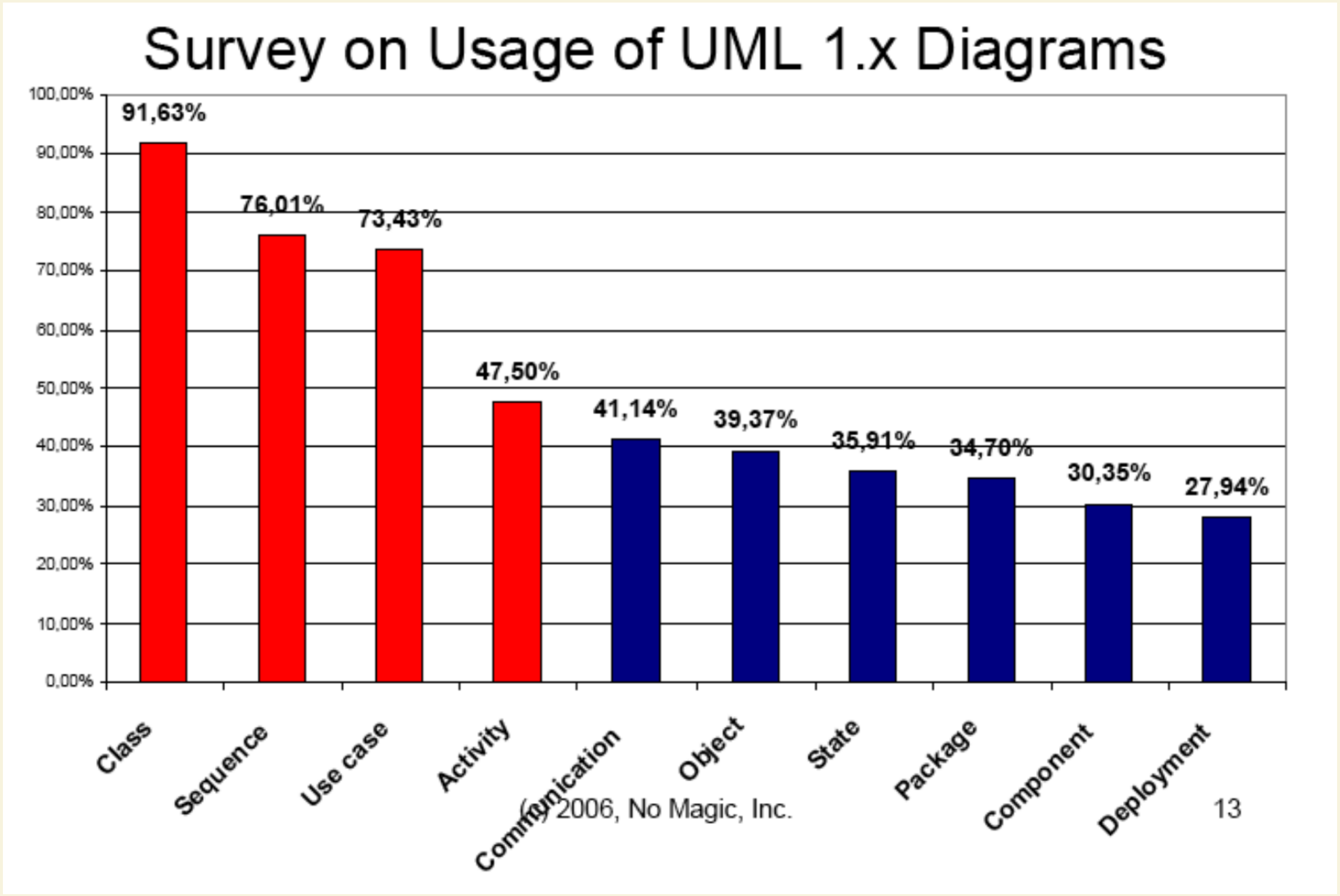
DIAGRAM TYPES



HOW DO PEOPLE USE UML ?

- Sketching tool
- Blueprint tool (Modeling)
- Code generation tool (Reverse Engineering)

USAGE


Survey on Usage of UML 1.x Diagrams
100,00%
90,00%
80,00%
70,00%
60,00%
50,00%
40,00%
30,00%
20,00%
10,00%
0,00%
91,63%
76,01%
73,43%
47,50%
41,14%
39,37%
35,91%
34,70%
30,35%
27,94%
Class
Sequence
Use case
Activity
Communication
Object
State
Package
Component
Deployment
(c) 2006, No Magic, Inc.
13

CLASS DIAGRAM

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. They also show properties and operations of a class.

WHAT ARE THE DIFFERNT PARTS OF THE CLASS DIAGRAM ?

- Properties
- Operations
- Relationships
- Notes
- Constraint rules
- Template (Parametrized classes)
- Responsibilities
- Static operations and attributes
- Abstract classes or operations

PROPERTIES

PROPERTIES

Correspond to the different variables of a class

PROPERTIES

Correspond to the different variables of a class

Form

```
visiblity name : type multiplicity = default
```

PROPERTIES

Correspond to the different variables of a class

Form

```
visiblity name : type multiplicity = default
```

Different `visibility` options

+	public
-	private
#	protected
~	package

PROPERTIES

Correspond to the different variables of a class

Form

```
visiblity name : type multiplicity = default
```

Different `visibility` options

+	public
-	private
#	protected
~	package

Different `multiplicity` options

1	Exactly one (Single value)
0..1	May or may not have one (Optional)
*	Many (Short for 0..*) (Multi valued)
x..y	To define an exact number ex. `2..4` or `2..*`

Code

```java
...
public class Book {
    public String title;
    private String author;
    protected int nbPages;
    String synopsis;
    public static int nbBooks;
    ...
}
```

Diagram

Code

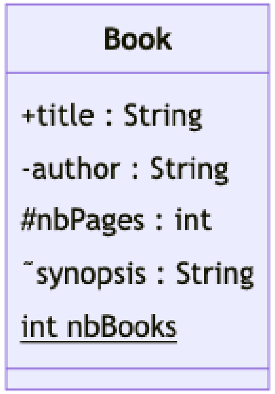
```java
...
public class Book {
    public String title;
    private String author;
    protected int nbPages;
    String synopsis;
    public static int nbBooks;
    ...
}
```

Diagram

Code

```java
...
public class Book {
    public String title;
    private String author;
    protected int nbPages;
    String synopsis;
    public static int nbBooks;
    ...
}
```

Diagram



OPERATIONS

Correspond to the different methods of a class

Form

```
visiblity name (parameter-list) : return-type
```

Normally, you don't show those operations that simply manipulate properties.

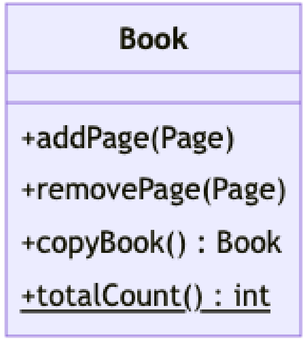
This is a conceptual model, you shouldn't use operations to specify the interface of a class, instead use them to indicate the principal responsibilities of that class.

Parameters are noted in the same way as attributes

Code

```java
...
public class Book {
    ...
    public void addPage(Page p) {
        ...
    }
    public void removePage(Page p) {
        ...
    }
    public Book copyBook() {
        ...
    }
    public static int totalCount() {
        ...
    }
```

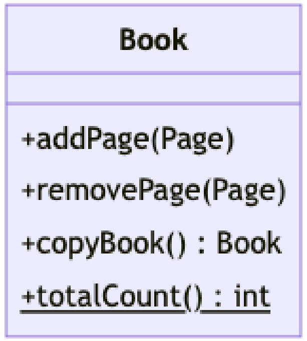
Diagram



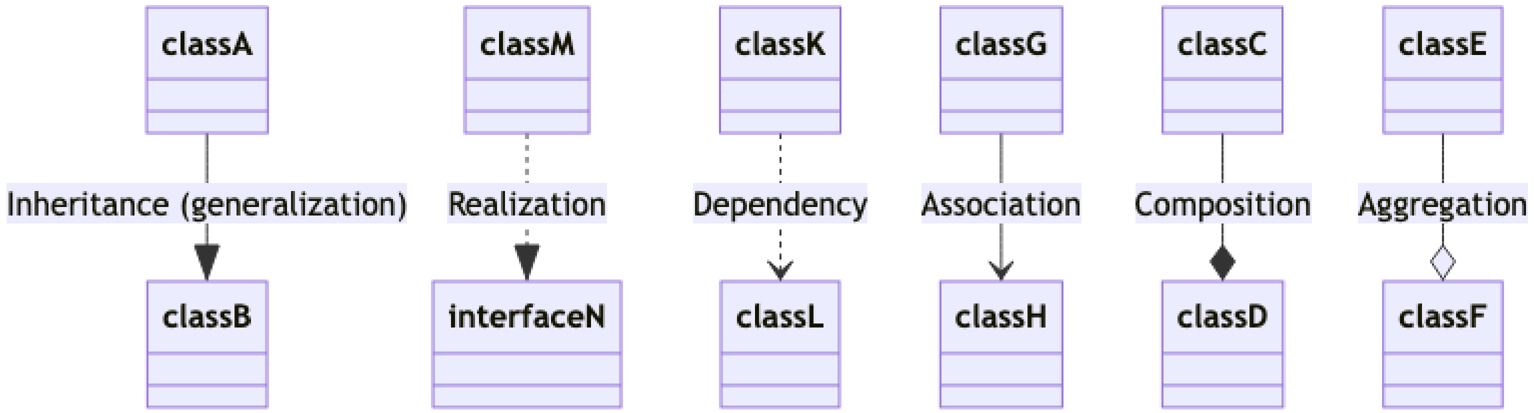
Code

```java
...
public class Book {
    ...
    public void addPage(Page p) {
        ...
    }
    public void removePage(Page p) {
        ...
    }
    public Book copyBook() {
        ...
    }
    public static int totalCount() {
        ...
    }
```

Diagram



RELATIONSHIPS



Type	Usage
Generalization	Implies an object that inherits some generalized fields from a supertype
Realization	Implies the implementation of an interface: denotes some responsibility which is not implemented by itself and the other entity that implements it
Dependency	Implies that an object accepts another object as a method parameter, instantiates, or uses another object
Association	Implies that one object has the other object as a property. Not to be confused with association classes
Composition	The object is owned by another object and its lifetime is bound the parent's lifetime, it is a special case of association

Dependency

Code

```java
public class Employee {
    private String name;
    private int baseSalary;
    private int overtime;

    public int getSalary(SalaryCalculator calc) {
        return calc.calculateSalary(baseSalary, overtime);
    }
}
public class SalaryCalculator {
    private float taxRate;
    public int calculate(int baseSalary, int overtime) {
        return ...;
    }
}
```

Diagram

Dependency

Code

```java
    public int getSalary(SalaryCalculator calc) {
        return calc.calculateSalary(baseSalary, overtime);
    }
}
public class SalaryCalculator {
    private float taxRate;
    public int calculate(int baseSalary, int overtime) {
        return ...;
    }
}
public static void main(String[] args) {
    Employee e1 = new Employee(...);
    SalaryCalculator calc = new SalaryCalculator(...);

    System.out.println(e.getSalary(calc));
```

Diagram

Dependency

Code

```java
        return calc.calculateSalary(baseSalary, overtime);
    }
}
public class SalaryCalculator {
    private float taxRate;
    public int calculate(int baseSalary, int overtime) {
        return ...;
    }
}
public static void main(String[] args) {
    Employee e1 = new Employee(...);
    SalaryCalculator calc = new SalaryCalculator(...);

    System.out.println(e.getSalary(calc));
}
```

Diagram

Dependency

Code

```java
public class Employee {
    private String name;
    private int baseSalary;
    private int overtime;

    public int getSalary(SalaryCalculator calc) {
        return calc.calculateSalary(baseSalary, overtime);
    }
}
public class SalaryCalculator {
    private float taxRate;
    public int calculate(int baseSalary, int overtime) {
        return ...;
    }
}
```

Diagram

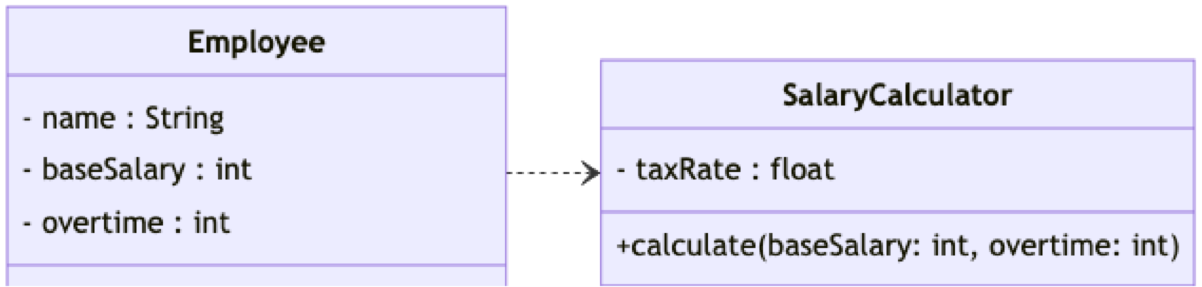
Dependency

Code

```java
public class Employee {
    private String name;
    private int baseSalary;
    private int overtime;

    public int getSalary(SalaryCalculator calc) {
        return calc.calculateSalary(baseSalary, overtime);
    }
}
public class SalaryCalculator {
    private float taxRate;
    public int calculate(int baseSalary, int overtime) {
        return ...;
    }
}
```

Diagram



Inheritance (generalization)

Code

```java
public class Customer {
    private String name;
    private int balance;
}
public class PersonalCustomer extends Customer {
    private String jobDescription;
}
public class CorporateCustomer extends Customer {
    private int nbEmployees;
}
```

Diagram

Inheritance (generalization)

Code

```java
public class Customer {
    private String name;
    private int balance;
}
public class PersonalCustomer extends Customer {
    private String jobDescription;
}
public class CorporateCustomer extends Customer {
    private int nbEmployees;
}
```

Diagram

Inheritance (generalization)

Code

```java
public class Customer {
    private String name;
    private int balance;
}
public class PersonalCustomer extends Customer {
    private String jobDescription;
}
public class CorporateCustomer extends Customer {
    private int nbEmployees;
}
```

Diagram

Inheritance (generalization)

Code

```java
public class Customer {
    private String name;
    private int balance;
}
public class PersonalCustomer extends Customer {
    private String jobDescription;
}
public class CorporateCustomer extends Customer {
    private int nbEmployees;
}
```

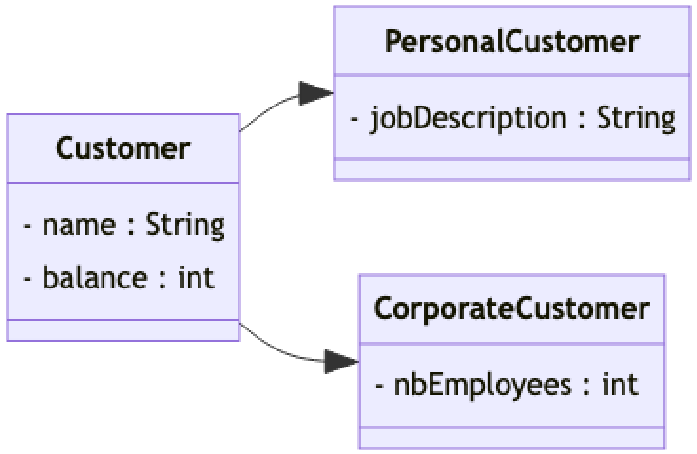
Diagram

Inheritance (generalization)

Code

```java
public class Customer {
    private String name;
    private int balance;
}
public class PersonalCustomer extends Customer {
    private String jobDescription;
}
public class CorporateCustomer extends Customer {
    private int nbEmployees;
}
```

Diagram



Association

Code

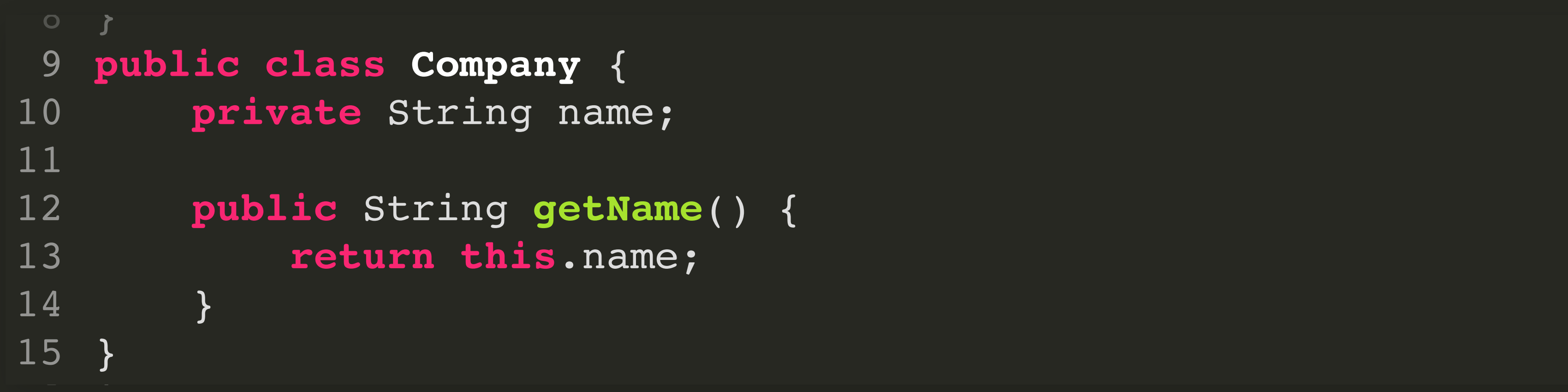
```
public class Employee {
    private String name;
    private Company company;

    public String toString() {
        return this.name + " works at " + this.company.name;
    }
}
```

Diagram

Association

Code



```java
}
public class Company {
    private String name;

    public String getName() {
        return this.name;
    }
}
```

Diagram

Association

Code

```java
public class Employee {
    private String name;
    private Company company;

    public String toString() {
        return this.name + " works at " + this.company.name;
    }
}
```

Diagram

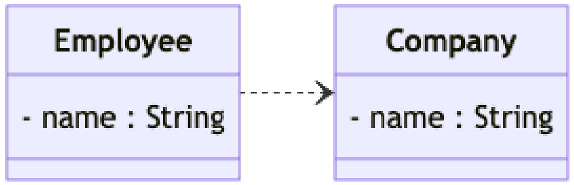
Association

Code

```java
public class Employee {
    private String name;
    private Company company;

    public String toString() {
        return this.name + " works at " + this.company.name;
    }
}
```

Diagram



Composition

Code

```java
public class University {
    private UniversityDepartment[] departments;
}
public class UniversityDepartment {
    private String name;
}
```

Diagram

Composition

Code

```java
public class University {
    private UniversityDepartment[] departments;
}
public class UniversityDepartment {
    private String name;
}
```

Diagram

Composition

Code

```java
public class University {
    private UniversityDepartment[] departments;
}
public class UniversityDepartment {
    private String name;
}
```

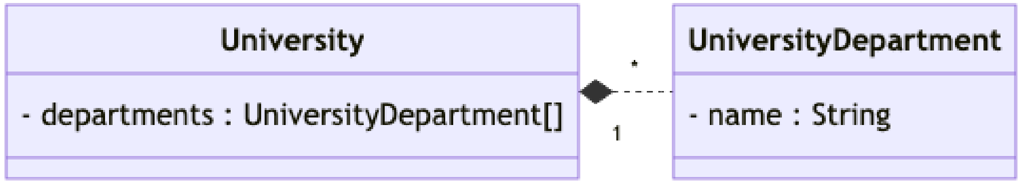
Diagram

Composition

Code

```java
public class University {
    private UniversityDepartment[] departments;
}
public class UniversityDepartment {
    private String name;
}
```

Diagram



Aggregation

Code

```java
public class UniversityDepartment {
    private Professor[] professors;
}
public class Professor {
    private String name;
}
```

Diagram

Aggregation

Code

```java
public class UniversityDepartment {
    private Professor[] professors;
}
public class Professor {
    private String name;
}
```

Diagram

Aggregation

Code

```java
public class UniversityDepartment {
    private Professor[] professors;
}
public class Professor {
    private String name;
}
```

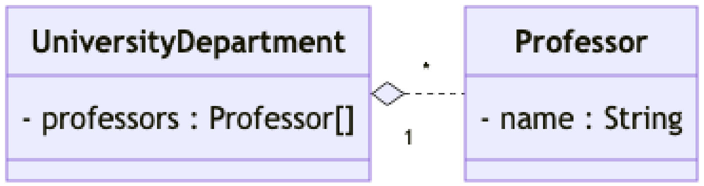
Diagram

Aggregation

Code

```java
public class UniversityDepartment {
    private Professor[] professors;
}
public class Professor {
    private String name;
}
```

Diagram



Realization

Code

```java
public interface Animal {
    public abstract void move();
    public abstract void eat();
    public abstract void sleep();
}
public class Kangaroo implements Animal {
    public void move() {
        ... // Code to jump
    }
    public void eat() {
        ... // Code to eat leaves
```

Diagram

Realization

Code

```java
public class Kangaroo implements Animal {
    public void move() {
        ... // Code to jump
    }
    public void eat() {
        ... // Code to eat leaves
    }
    public void sleep() {
        ... // Code to sleep infrequently (15h on 4 days)
    }
}
public class Elephant implements Animal {
```

Diagram

Realization

Code

```java
public class Elephant implements Animal {
    public void move() {
        ... // Code to move slowly
    }
    public void eat() {
        ... // Code to eat grass
    }
    public void sleep() {
        ...// Code to sleep for 4 hours
    }
}
```

Diagram

Realization

Code

```java
public interface Animal {
    public abstract void move();
    public abstract void eat();
    public abstract void sleep();
}
public class Kangaroo implements Animal {
    public void move() {
        ... // Code to jump
    }
    public void eat() {
        ... // Code to eat leaves
```

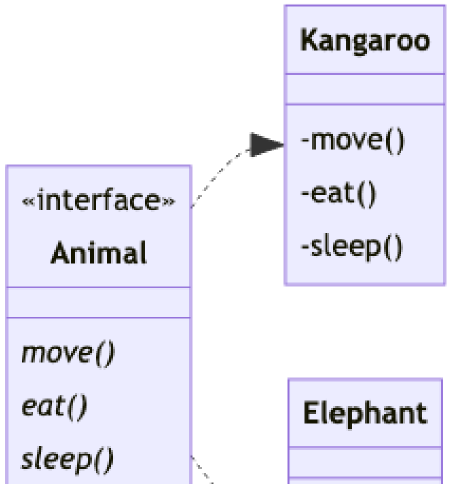
Diagram

Realization

Code

```java
public interface Animal {
    public abstract void move();
    public abstract void eat();
    public abstract void sleep();
}
public class Kangaroo implements Animal {
    public void move() {
        ... // Code to jump
    }
    public void eat() {
        ... // Code to eat leaves
```

Diagram



Notes

Provide comments on the diagram, can be linked by a dashed line or not.

Code

```
public class Car {
}
```

Diagram

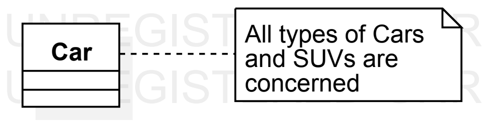
Notes

Provide comments on the diagram, can be linked by a dashed line or not.

Code

```java
public class Car {
}
```

Diagram



Constraint rules

Must be places in {} and they can be used to indicate any type of constraint that is not apparent by the properties, operations, associations and generalizations

Examples

{readOnly}

{ordered}

{nonunique}

...

Code

```
public class Order {
    private OrderLine[] lines;
}
```

Diagram

Constraint rules

Must be places in {} and they can be used to indicate any type of constraint that is not apparent by the properties, operations, associations and generalizations

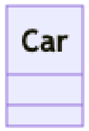
Examples

{readOnly}
{ordered}
{nonunique}
...

Code

```java
public class Order {
    private OrderLine[] lines;
}
```

Diagram



Parametrized classes

Used to describe generic classes

Code

```java
public class Student {
    private String name;
    private int grade;
}
public class Employee {
    private String name;
    private int salary;
}
```

Diagram

Parametrized classes

Used to describe generic classes

Code

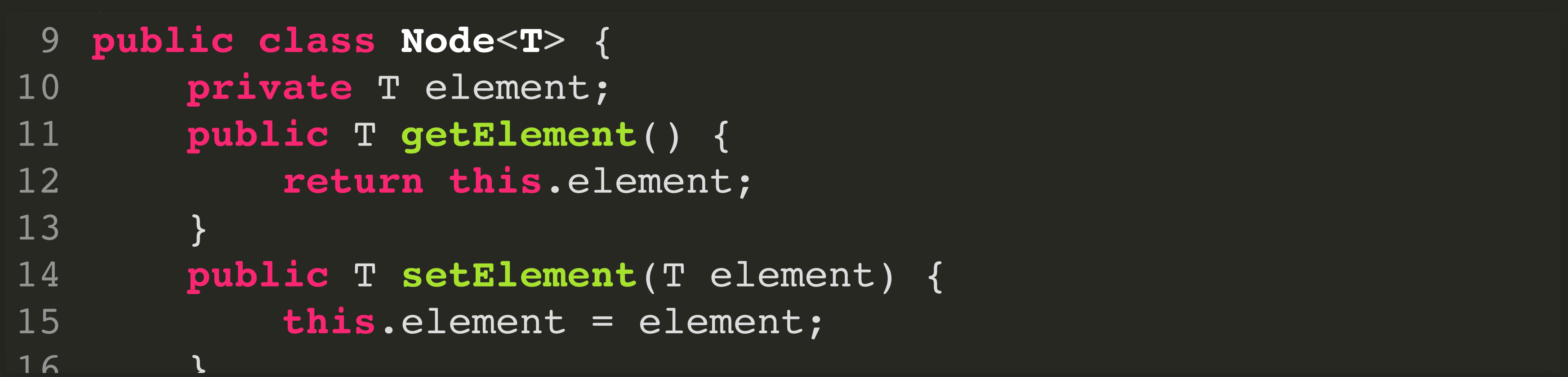
```
    private int grade;
}
public class Employee {
    private String name;
    private int salary;
}
public class Node<T> {
    private T element;
```

Diagram

Parametrized classes

Used to describe generic classes

Code



```java
public class Node<T> {
    private T element;
    public T getElement() {
        return this.element;
    }
    public T setElement(T element) {
        this.element = element;
    }
```

Diagram

Parametrized classes

Used to describe generic classes

Code

```
public static void main (String[] args) {
    Student s = new Student(...);
    Node<Student> n = new Node<>();
    n.setElement(s);

    Employee e = new Employee(...);
    Node<Employee> n2 = new Node<>();
    n2.setElement(e);
```

Diagram

Parametrized classes

Used to describe generic classes

Code

```java
public class Student {
    private String name;
    private int grade;
}
public class Employee {
    private String name;
    private int salary;
}
```

Diagram

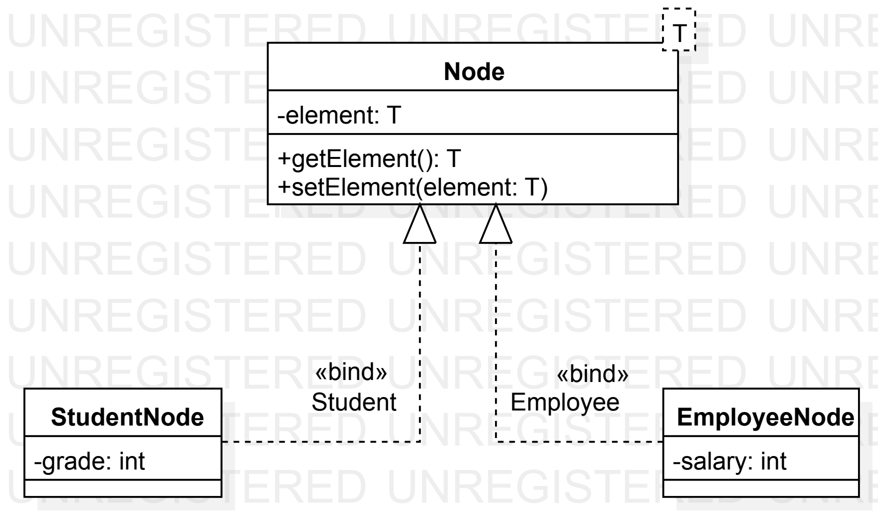
Parametrized classes

Used to describe generic classes

Code

```java
public class Student {
     private String name;
     private int grade;
}
public class Employee {
     private String name;
     private int salary;
}
```

Diagram



Static operations and attributes

Code

```java
public class Student {
    private String prefixCode = 'S';
    public static String getPrefixCode() {
        return prefixCode;
    }
}
public class Employee {
```

Diagram

Static operations and attributes

Code

```java
    }
}
public class Employee {
    private String prefixCode = 'E';
    public static String getPrefixCode() {
        return prefixCode;
    }
}
```

Diagram

Static operations and attributes

Code

```
public class Student {
    private String prefixCode = 'S';
    public static String getPrefixCode() {
        return prefixCode;
    }
}
public class Employee {
    private String prefixCode = 'E';
```

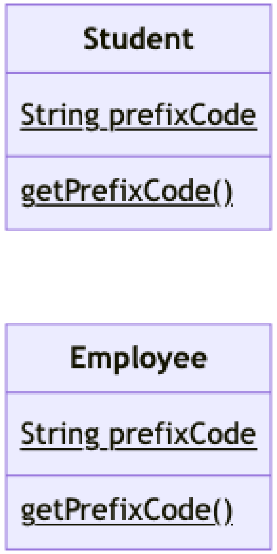
Diagram

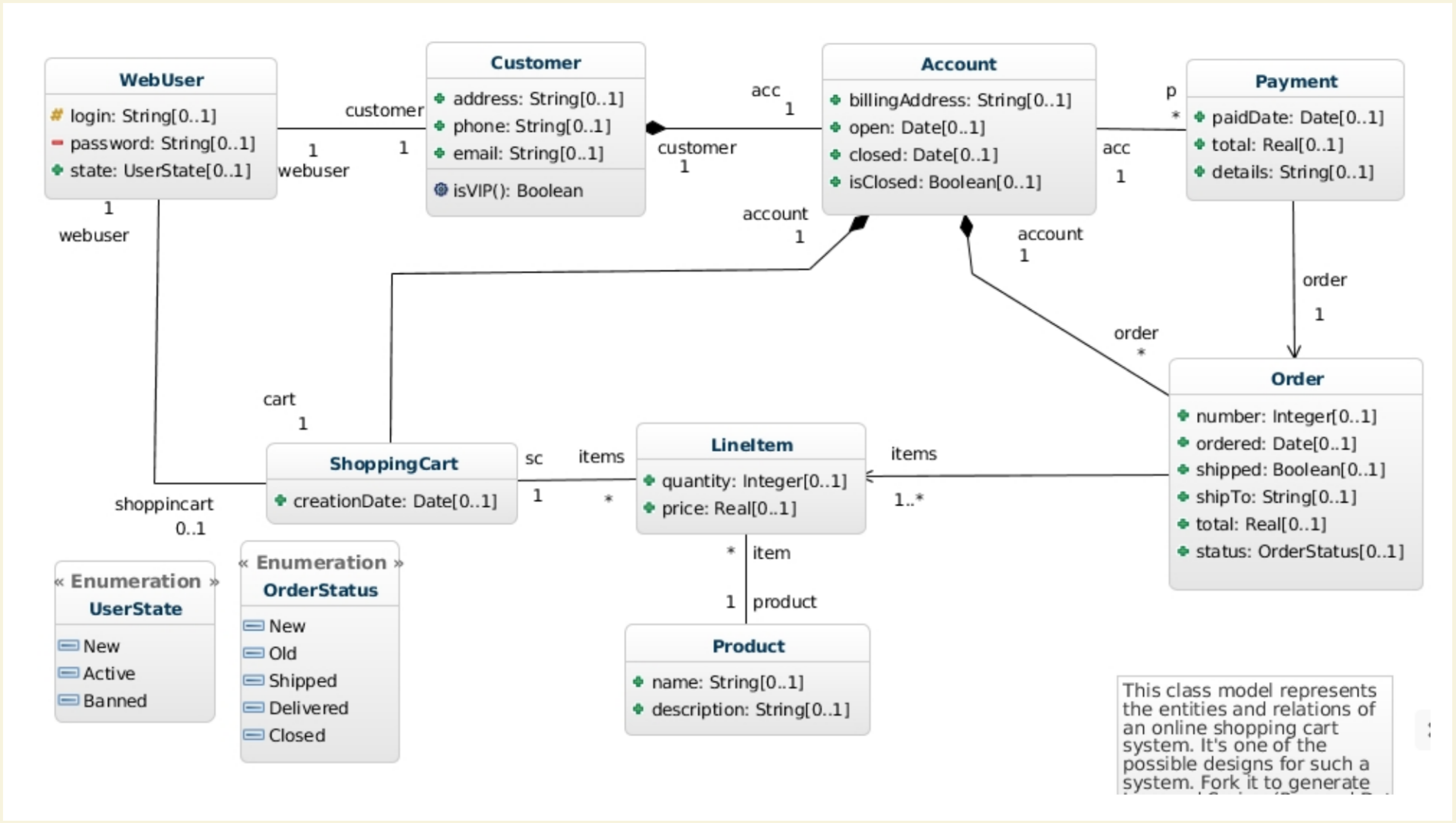
Static operations and attributes

Code

```java
public class Student {
    private String prefixCode = 'S';
    public static String getPrefixCode() {
        return prefixCode;
    }
}
public class Employee {
    private String prefixCode = 'E';
```

Diagram





This class model represents the entities and relations of an online shopping cart system. It's one of the possible designs for such a system. Fork it to generate

SEQUENCE DIAGRAM

SEQUENCE DIAGRAM

It is an interaction diagram : describes how groups of objects collaborate in some behaviour. How do the objects interact.

SEQUENCE DIAGRAM

It is an interaction diagram : describes how groups of objects collaborate in some behaviour. How do the objects interact.

Captures behavior of a single scenario. It show a number of example objects and the messages that are passed between these objects within the use case.

SEQUENCE DIAGRAM

It is an interaction diagram : describes how groups of objects collaborate in some behaviour. How do the objects interact.

Captures behavior of a single scenario. It show a number of example objects and the messages that are passed between these objects within the use case.

They show clearly what objects are doing what calculations and delegating others

WHAT ARE THE DIFFERENT PARTS OF THE SEQUENCE DIAGRAM ?

- Participants
- Messages
- Return arrows
- Lifelines
- Activation bars
- Creation / deletion of participants
- Interaction frames
- Guards
- Synchronous / Asynchronous calls

SCENARIO

We have an order and are going to invoke a command on it to calculate its price.
To do that, the order needs to look at all the line items on the order and determine their prices,
which are based on the pricing rules of the order line's products.
Having done that for all the line items, the order then needs to compute an overall discount, which
is based on rules tied to the customer.

PARTICIPANTS

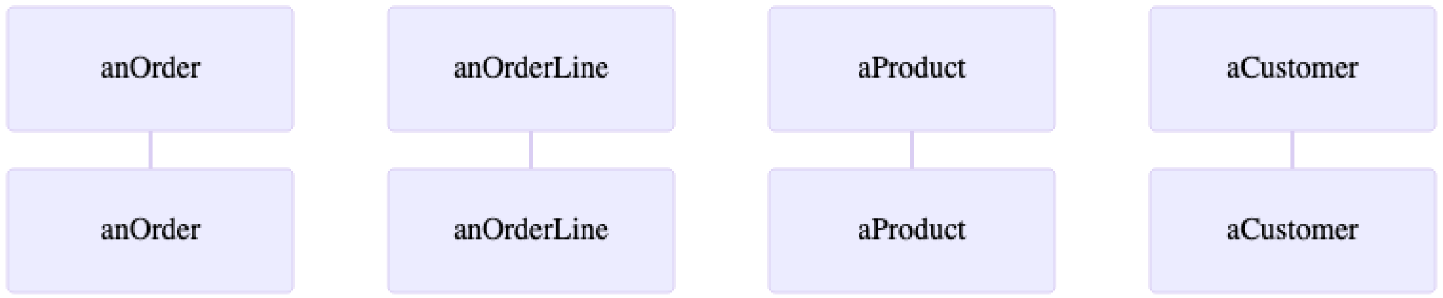
We start by adding the participants & their lifelines

Diagram

PARTICIPANTS

We start by adding the participants & their lifelines

Diagram



MESSAGES

Then we add the messages and their return arrows

Diagram

MESSAGES

Then we add the messages and their return arrows

Diagram

ACTIVATION BARS

Then we add activation bars

Diagram

ACTIVATION BARS

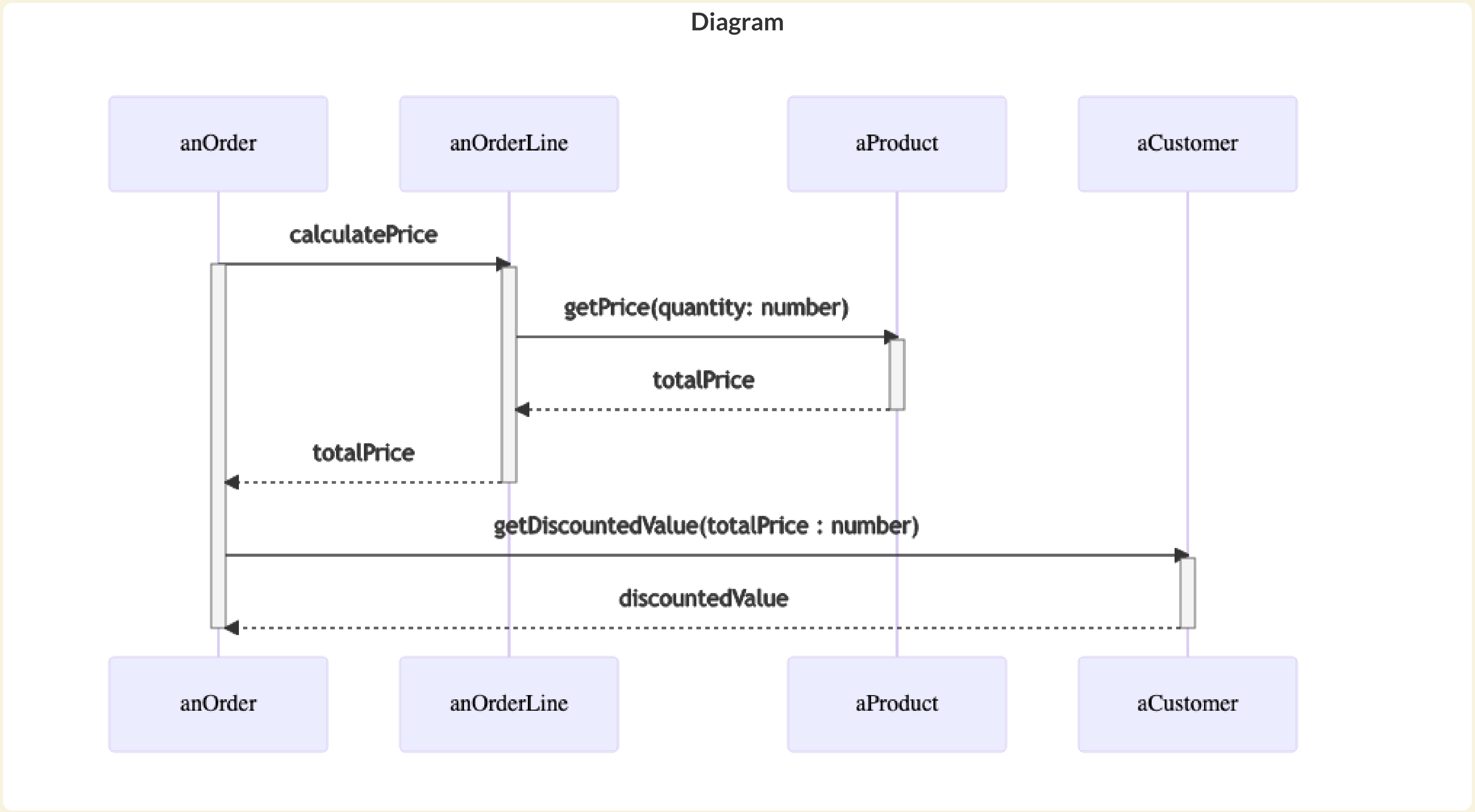
Then we add activation bars

Diagram

ANOTHER WAY OF REPRESENTING THE SAME SCENARIO

Diagram

ANOTHER WAY OF REPRESENTING THE SAME SCENARIO



CREATION & DELETION

Diagram

CREATION & DELETION

Diagram

ACTIVITY DIAGRAM

ACTIVITY DIAGRAM

Very similary to flow charts, the only difference being that the activity diagrams can describe parallel processes

ACTIVITY DIAGRAM

Very similary to flow charts, the only difference being that the activity diagrams can describe parallel processes

Describe different workflows

WHAT ARE THE DIFFERENT PARTS OF THE ACTIVITY DIAGRAM ?

- Start node
- Action
- Decision node
- Merger node
- Parallel node (concurrent flows)
- Final node

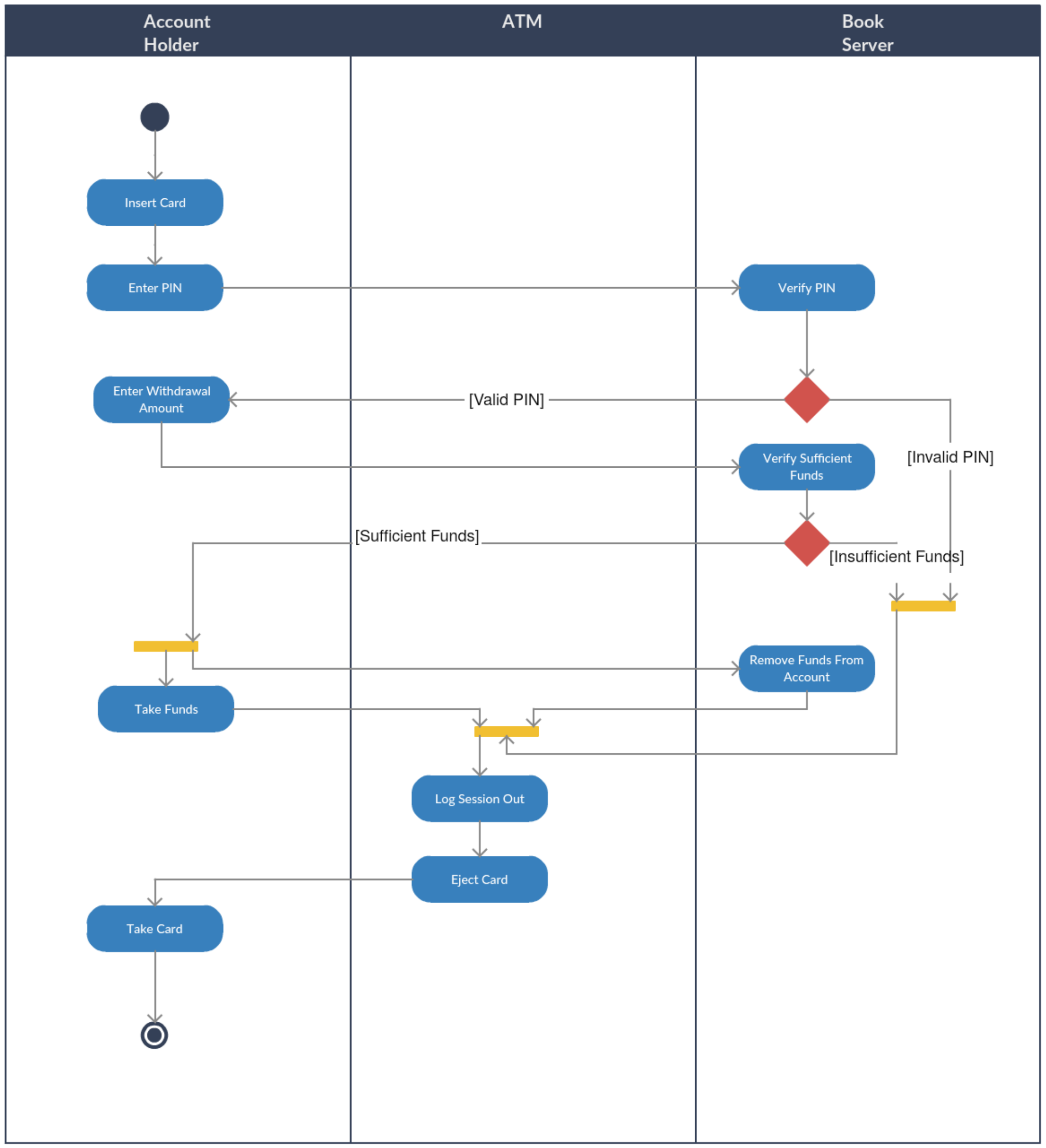
EXAMPLE TO CLARIFY THE DIFFERENT PARTS

Diagram

EXAMPLE TO CLARIFY THE DIFFERENT PARTS

Diagram

ATM SYSTEM for ABC BANK



USE CASE DIAGRAM

USE CASE DIAGRAM

Describes the functional requirements of the system: the functionality from the user's POV.

USE CASE DIAGRAM

Describes the functional requirements of the system: the functionality from the user's POV.

Communicate high-level features of the app

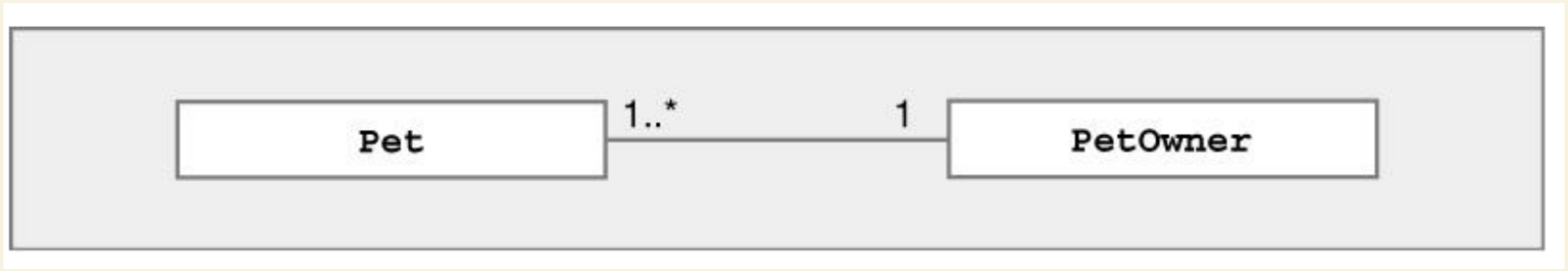
WHAT ARE THE DIFFERENT PARTS OF THE ACTIVITY DIAGRAM ?

- Actors
- Actions (features)
- App Frame
- Links

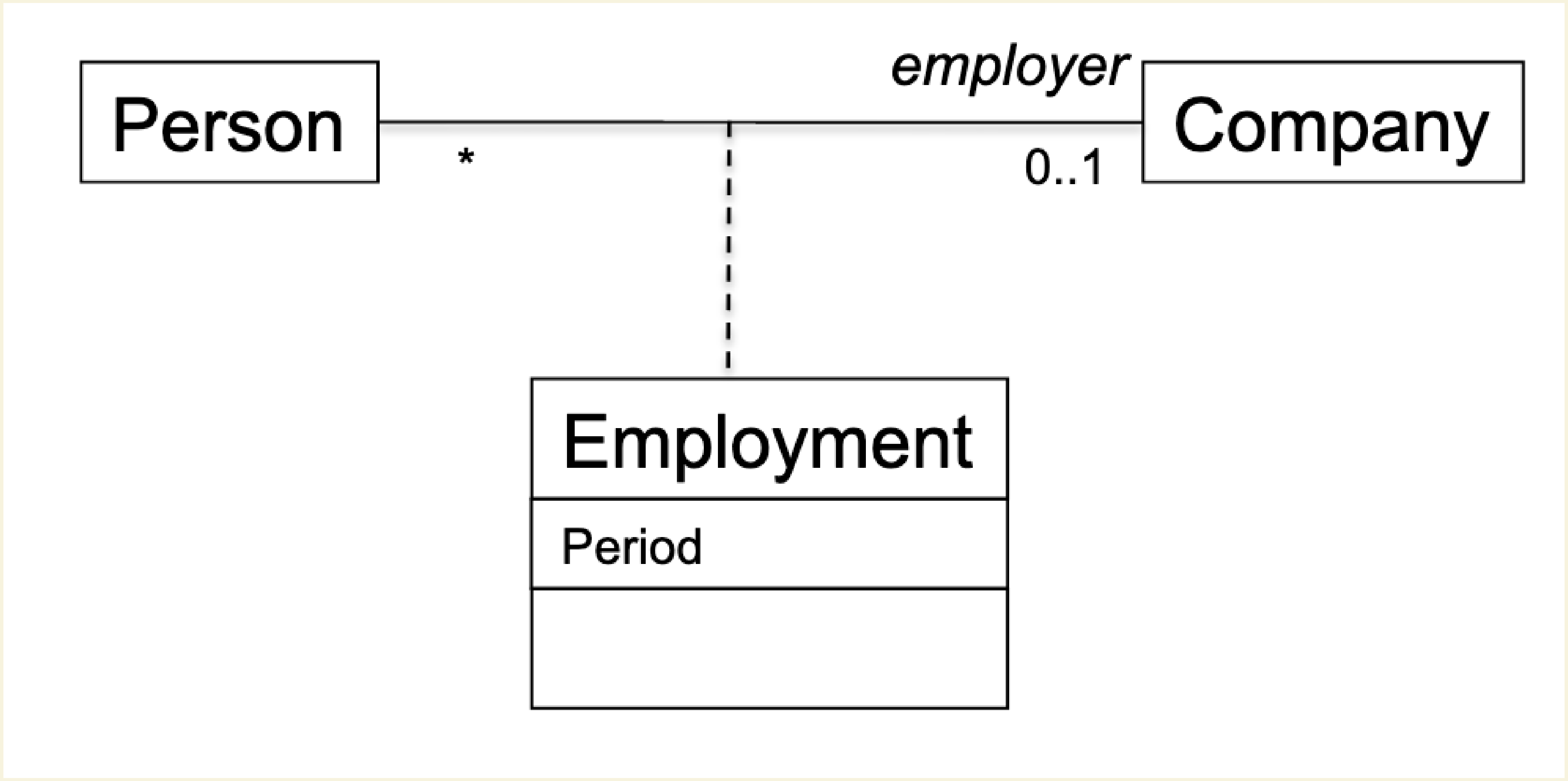
EXAMPLE TO EXPLORE THE DIFFERENT PARTS

EXERCISES

READ AND UNDERSTAND THIS UML DIAGRAM

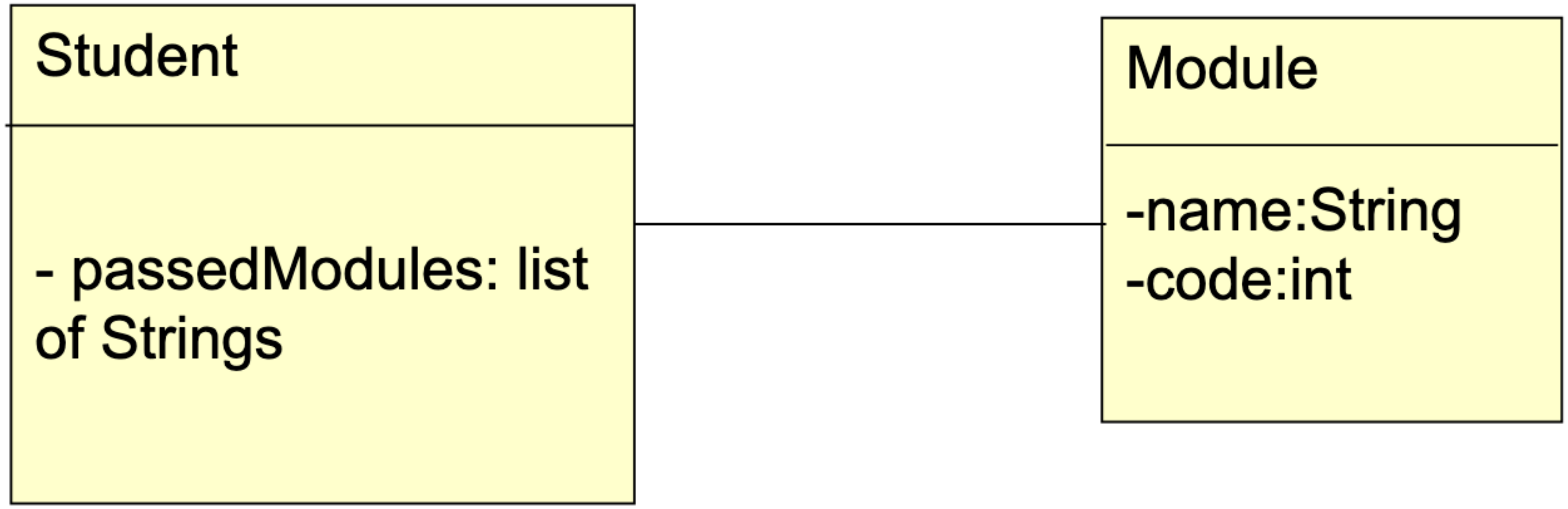


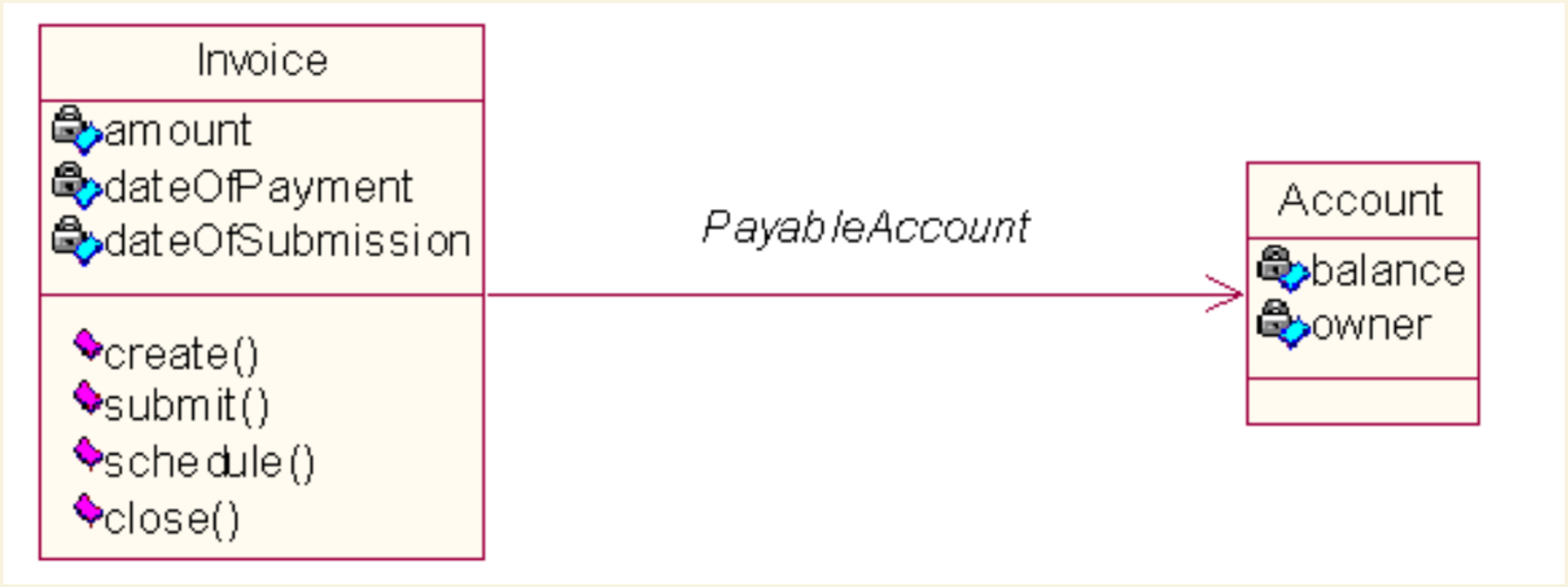
CLASS DIAGRAMS



Which sentences are coherent with this model?

- Right or wrong?



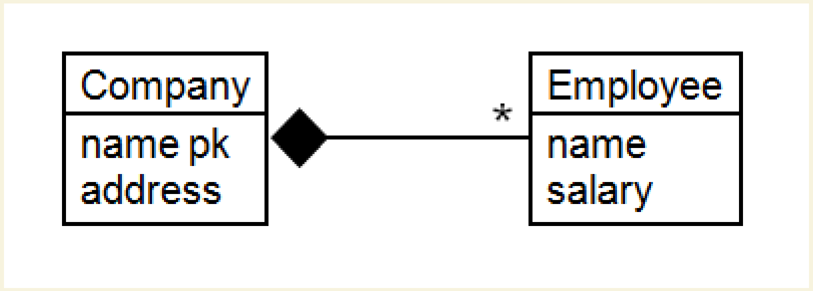


The arrow between the two classes indicates:

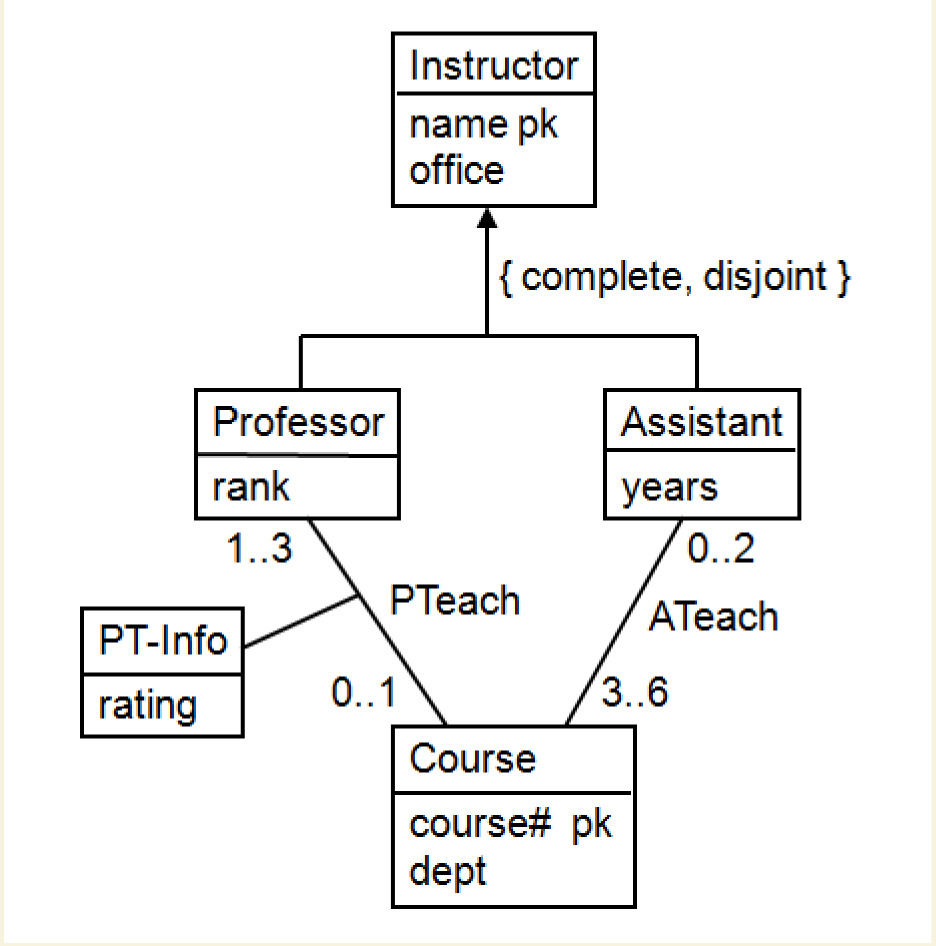
1. Inheritance
2. Association
3. Dependency
4. Sending a message

AGGREGATION OR COMPOSITION ?

- Building and rooms
- Course and lessons
- TV channel and programs
- Parliament and members
- Sky and stars
- Country and cities
- City and buildings
- Wood and trees



- No two companies can have the same name
- No two employees can have the same name
- No two companies can be at the same address
- No two employees can work at the same address
- Each employee works for at least one company
- No employees work for more than one company
- Each company has at least one employee
- Two employees with the same name cannot work for the same company
- Two employees with the same name cannot work for different companies



- According to the diagram, what are the minimum and maximum total number of instructors for a given course?
- According to the diagram, what is the minimum and maximum teaching load (number of courses) for professors? For assistants?

IMAGINE SOME AGGREGATION OR COMPOSITION RELATIONSHIPS AMONG THE FOLLOWING CLASSES AND DRAW A CORRESPONDING CLASS DIAGRAM

- Employee
- Manager
- Office
- Department

CLASS DIAGRAM EXERCISE

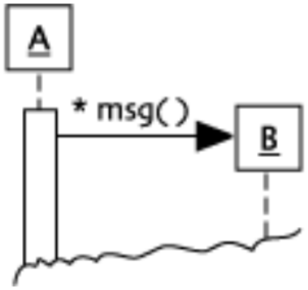
- A house may have any number of pets living in it
- The two possible types of pets that can live in a house are dogs and cats
- Each dog or cat has a name
- An animal’s house is its one and only home
- You can tell an animal to make noise

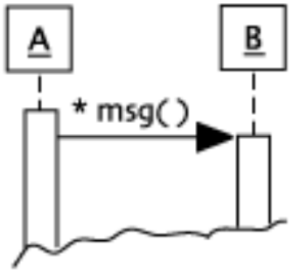
UML MODELING EXERCISE

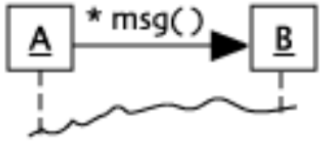
- Draw a UML class diagram that models the relationships between the following classes: Mall, Store, Sales Person, Department, Manager, Merchandise, Store Catalog, Store Website, and Customer.

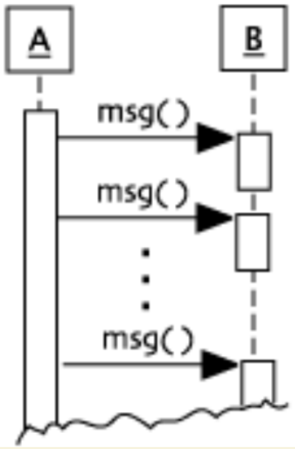
- Which one of the following fragments of sequence diagram represents the action: "Object A sends to object B several messages named *msg*"?

a) b) c) d)





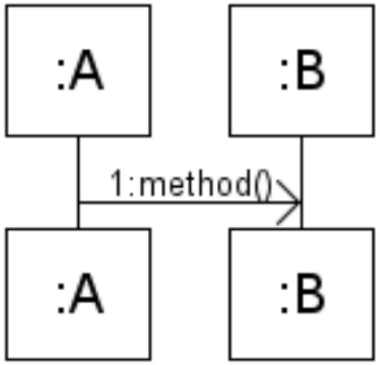




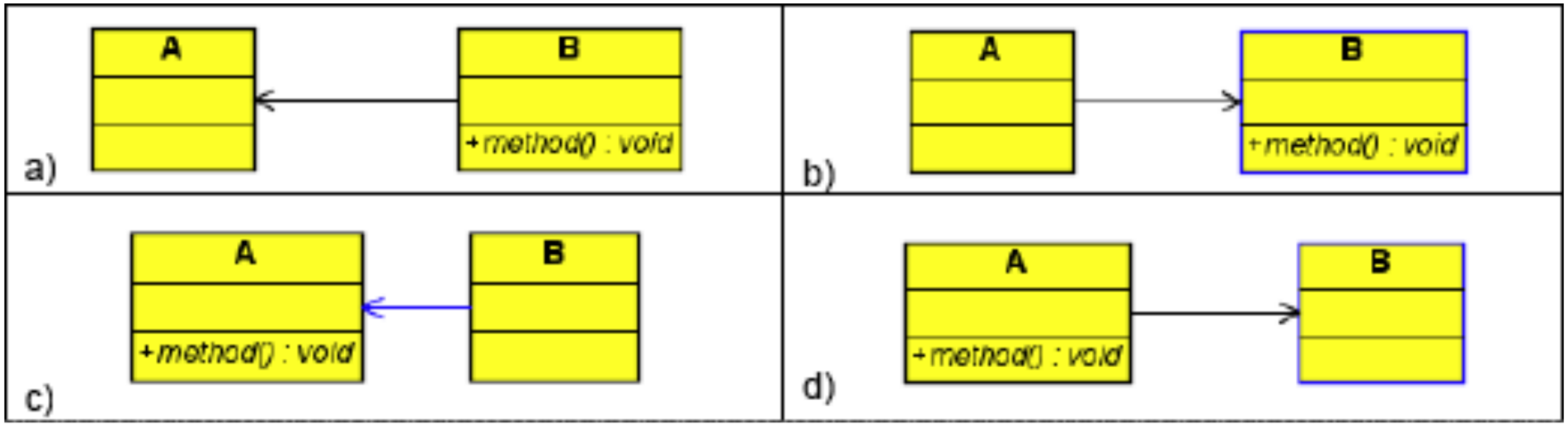
UML MODELING EXERCISE

- Create a UML **sequence diagram** for the Withdraw Cash success use case for the ATM system
- Is it possible to include extension 3a (cash not available) in the same sequence diagram? Why or why not?
- Create a UML **activity diagram** for the Withdraw Cash use case

On sequence diagrams

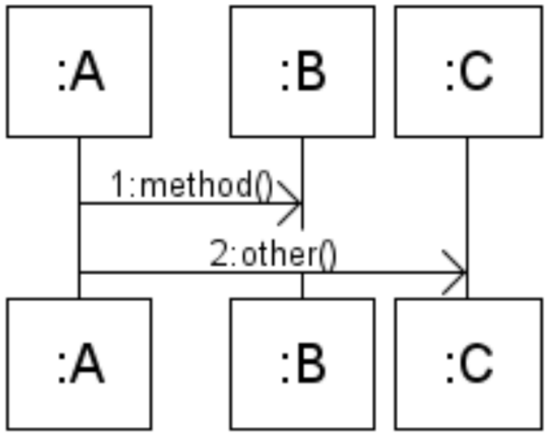


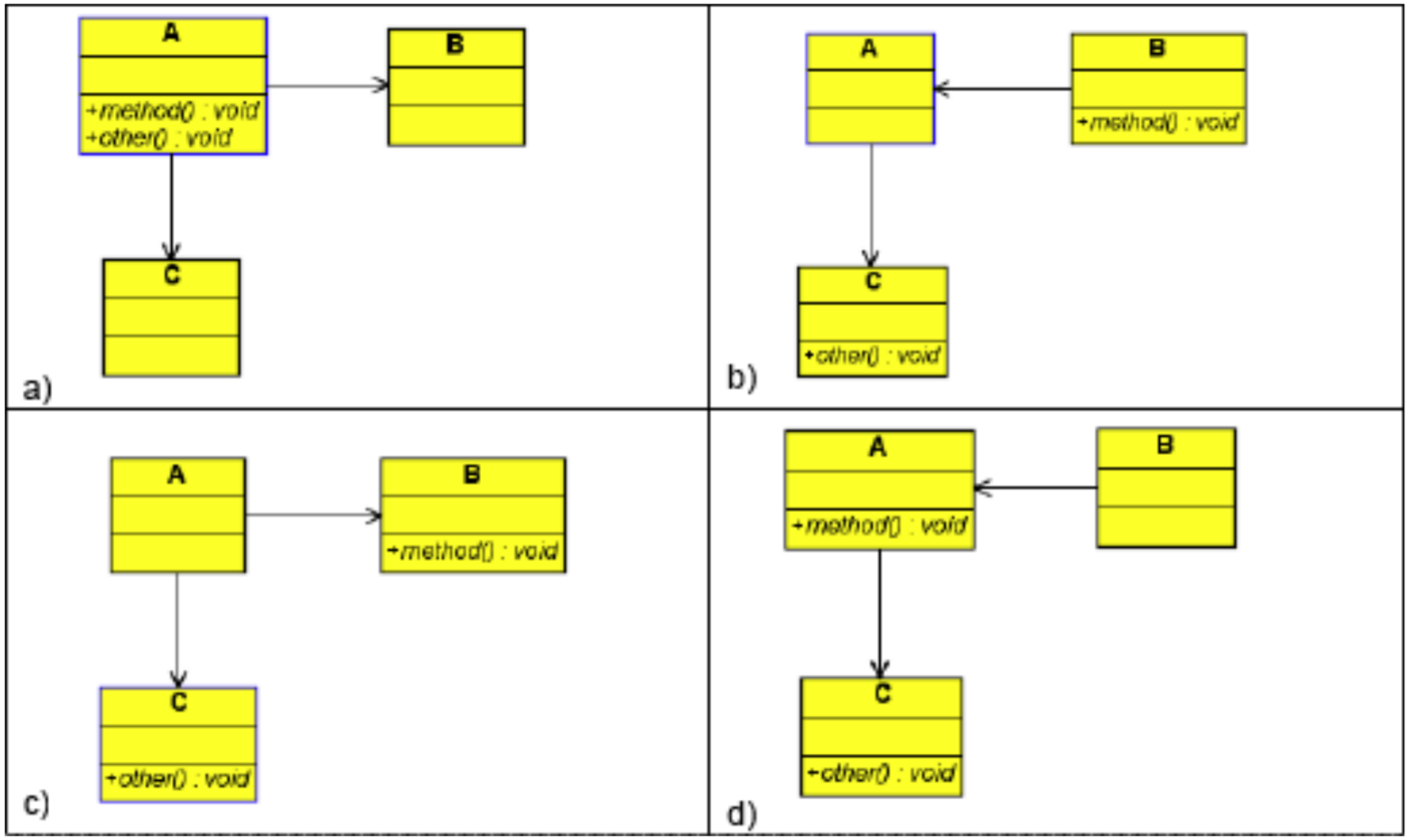
Given the sequence diagram on right, which class diagram is consistent?



On sequence diagrams

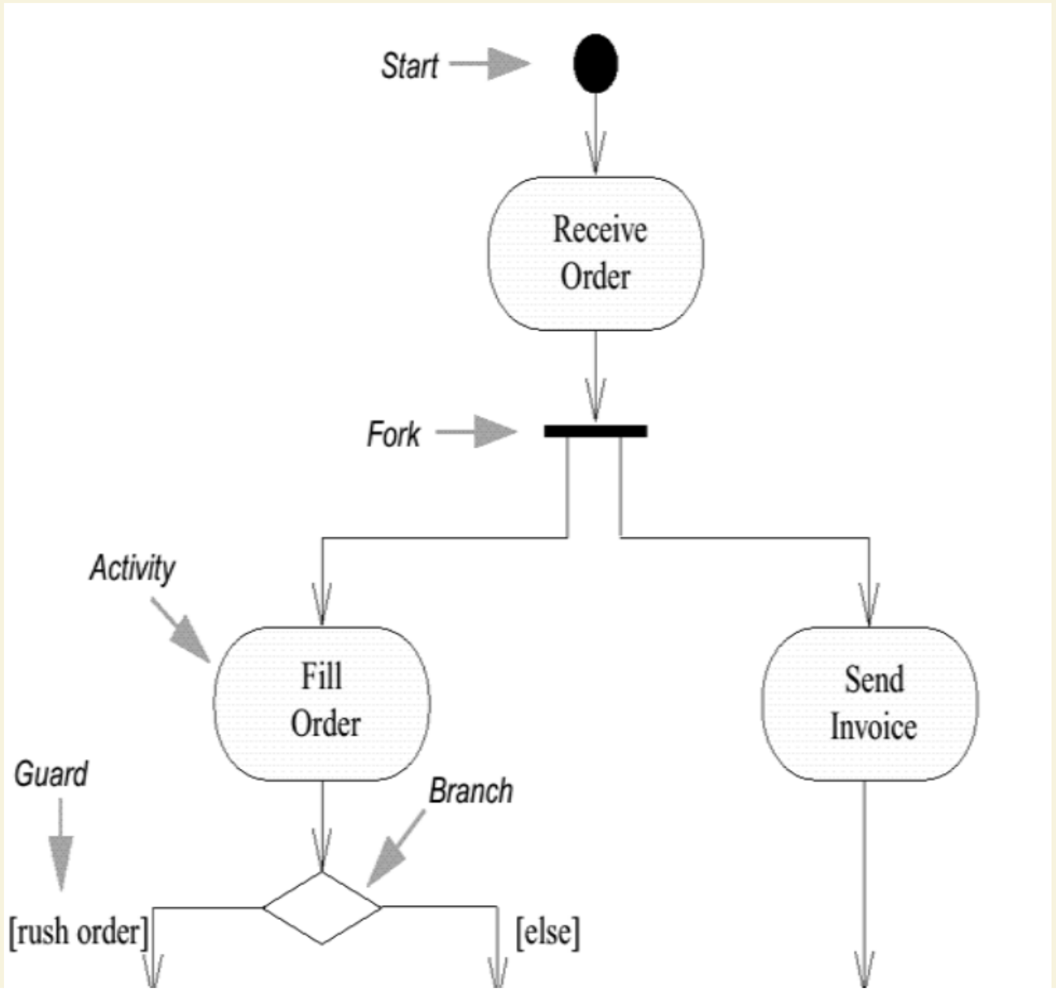
Given the sequence diagram on right, which class diagram is consistent?





SEQUENCE DIAGRAM EXERCISE

- Draw a sequence diagram showing how a customer interacts with a travel agency, a station and a train to reach some destination
- Draw a sequence diagram to show how a user prints a document on a printer, and a counter keeps a count of printed pages



In this diagram ?

- Fill Order is executed before Receive Payment
- Overnight Delivery is executed in parallel with Regular Delivery
- Close Order is executed after Receive

USE CASE DIAGRAM EXERCISE

Propose a use case diagram for an ATM machine for withdrawing cash. Make the use case simple yet informative; only include the major features.

USE CASE DIAGRAM

Propose a use case diagram for a vending machine that sells beverages and snacks..

THANK YOU