برنامه نویسی پیشرفته

رفع اشکال: جلسه ۶

Software Engineering , coupling and cohesion
UML
SQLite

Software Engineering Essentials

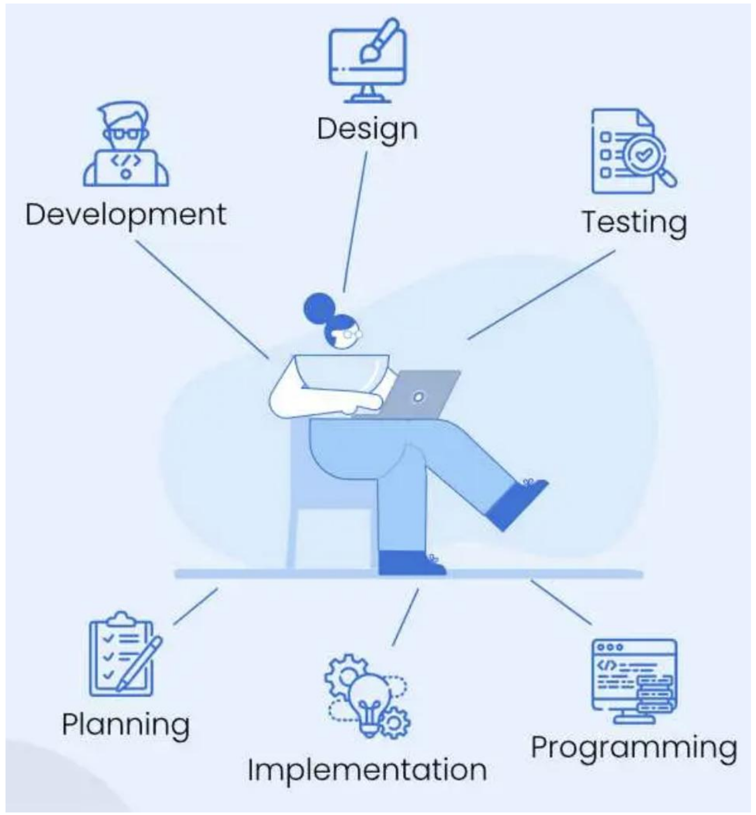
Definition : Systematic approach to the development, operation, maintenance of software.

Key Challenges:

- Managing complexity (large-scale projects)
- Ensuring maintainability and extensibility
- Handling continuous change requests

Advanced Java Context:

•Designing classes with best practices ensures flexible, robust code.



COUPLING: DEFINITION & TYPES

What is Coupling?

- The level of interdependence between modules/classes.

Tight Coupling:

- Classes share many details or directly access each other's internals (e.g., public fields).

Loose Coupling:

- Classes interact through clear, minimal interfaces.
- Example: Using getters/setters or service interfaces.

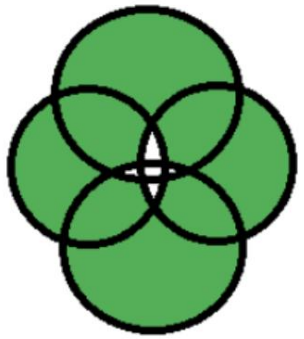
```
// ClassA exposes its internal state publicly
public class ClassA {
    public int value; // BAD: public field

    public ClassA(int value) {
        this.value = value;
    }
}

// ClassB directly manipulates ClassA's internal field

public class ClassB {
    public void doubleValue(ClassA a) {
        a.value = a.value * 2; // tightly coupled to ClassA's
internals
    }
}
```

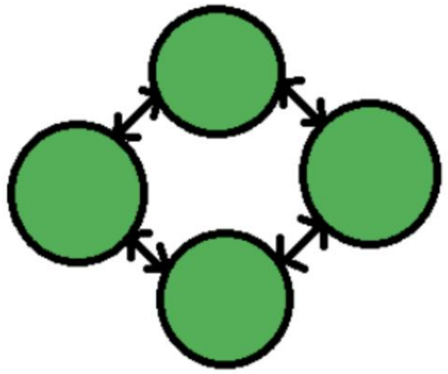
PROBLEMS WITH TIGHT COUPLING

Difficult Maintenance: Changes in one class force changes in dependent classes.

Reduced Reusability: Hard to reuse classes that rely on specific implementations.

Complex Testing: Requires setting up the entire chain of dependencies for a single test.





COHESION: DEFINITION & LEVELS

- **What is Cohesion?**
  - How strongly-related and focused the responsibilities of a single class/method are.
- **High Cohesion:**
  - A class is responsible for a single concept.
- **Low Cohesion** :
  - A single class handles multiple unrelated tasks.
  - Harder to read, maintain, and reuse.

```
// Low Cohesion Example
public class EmployeeUtils {
    public void generatePaySlip() { ... }
    public void printEmployeeReport() { ... }
    public void sendEmailToEmployee() { ... }
    public void calculateTaxDeductions() { ... }
}

// High Cohesion Example
public class PayrollService {
    public PaySlip generatePaySlip(Employee emp) { ... }
}
public class EmailService {
    public void sendEmail(String recipient, String subject, String
body) { ... }
}
```

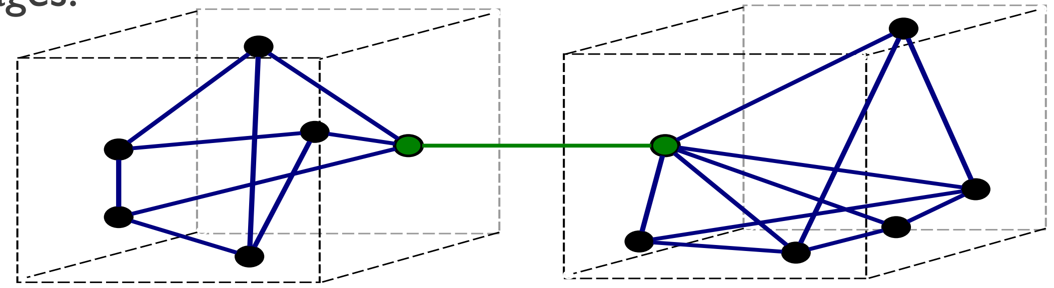
ACHIEVING LOOSE COUPLING & HIGH COHESION

Encapsulation: Keep data private; expose behavior via methods.

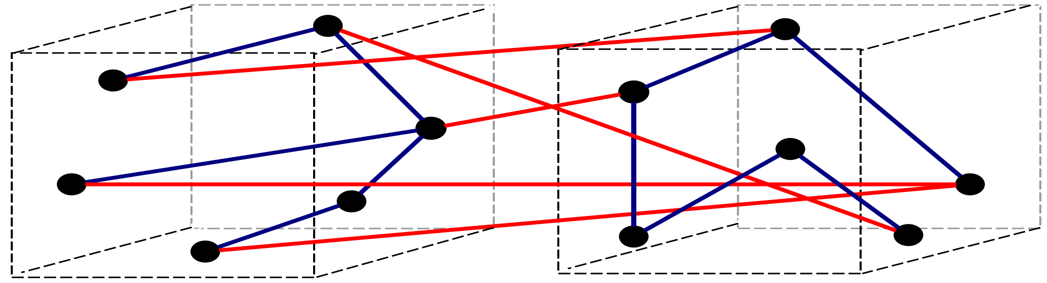
Single Responsibility Principle (SRP): Each class should have one reason to change.

Dependency Injection / Interfaces: Classes depend on abstractions, not concretions.

Modular Design: Group related functionalities into separate packages.



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

QUESTION: WHAT IS THE BEST IMPROVEMENT?

You are tasked with refactoring a PaymentProcessor class in a legacy Java application. Currently, the class does all the following:

- Connects to multiple payment gateways (e.g., Stripe, PayPal)
- Validates user credit card inputs
- Logs every transaction to a local file
- Updates the customer's transaction history in the database

A. Move all the functionality into a TransactionManager class and use it across the system for reuse.

B. Split the responsibilities into PaymentGatewayService, InputValidator, TransactionLogger, and CustomerHistoryUpdater classes, and inject them into PaymentProcessor via interfaces.

C. Keep the structure as-is but mark helper methods as private to reduce visibility and improve encapsulation.

D. Move the payment gateway code to a utility class with static methods to allow easier reuse without object instantiation.

ANSWER : B

Option B supports **high cohesion** by giving each class a single, focused responsibility.
It also promotes **loose coupling** by using dependency injection through interfaces, making each component independently testable and replaceable.

Option A shifts the problem into another class (still low cohesion).
Option C improves encapsulation but doesn’t address coupling or cohesion properly.
Option D allows reuse, but static utility classes limit flexibility and don’t support polymorphism or dependency inversion.

INTRODUCTION TO UML

UML (Unified Modeling Language) is a standardized visual language for:

- Modeling software systems before coding
- Documenting structure and behavior
- Communicating design clearly between team members

Think of it as the *blueprint of your codebase* (the same way architects use blueprints before building)

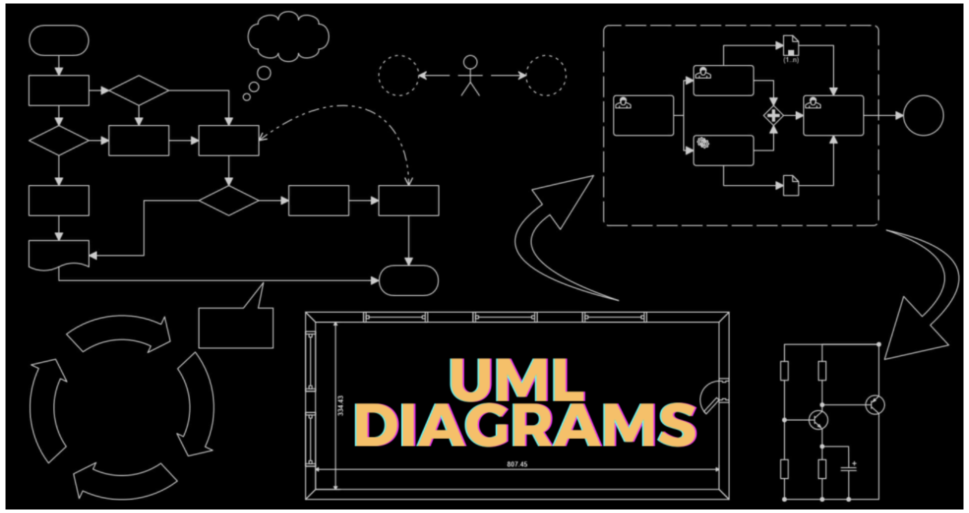
Why Should Java Developers Care?

- Design Clarity
- Standardization
- Documentation Tool
- Design Validation

Types of UML Diagrams

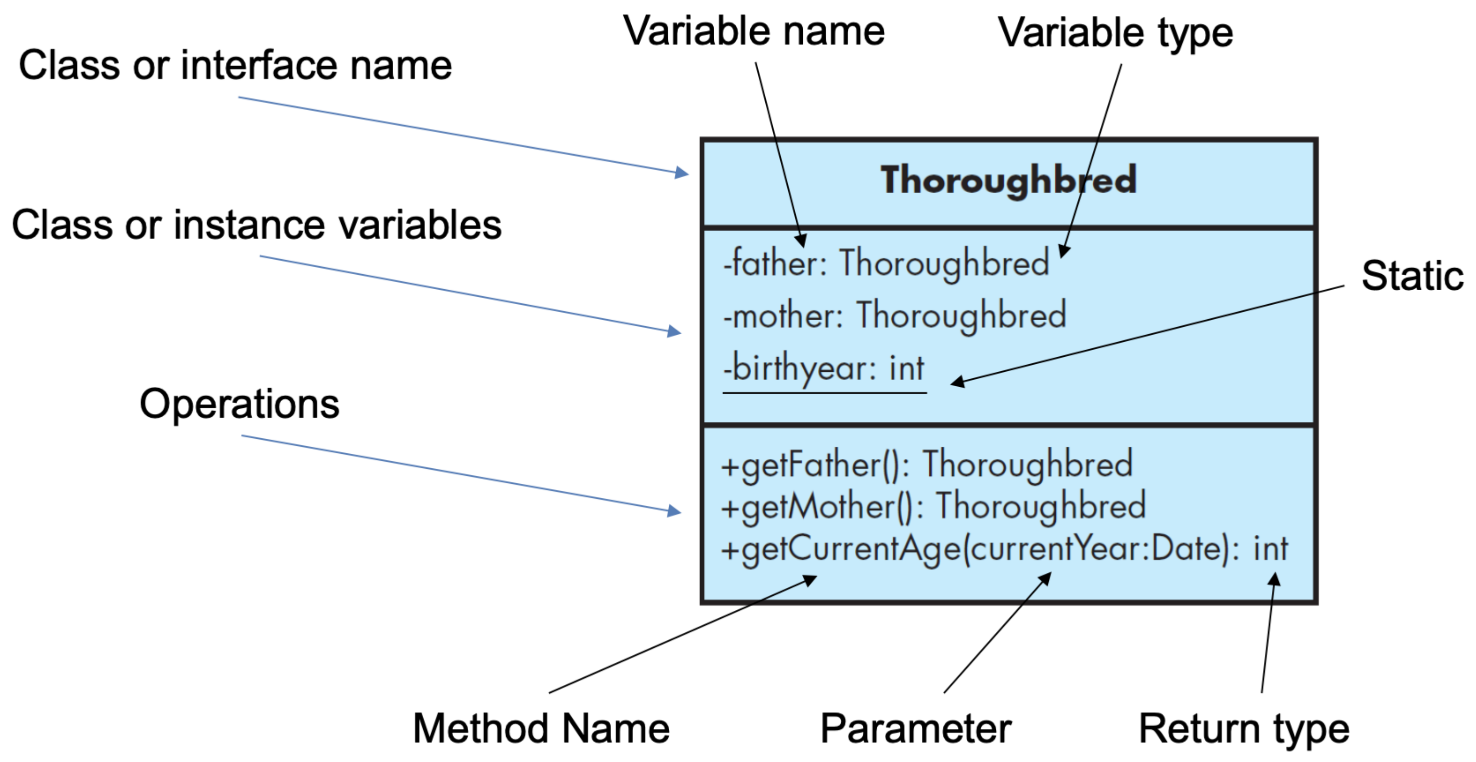
CLASS , USE CASE , SEQUENCE , ACTIVITY

we'll focus primarily on Class Diagrams in this course.



CLASS DIAGRAM ELEMENTS

- **Components:**
  - **1-Classes:** Typically shown as boxes with 3 compartments (Name, Attributes, Methods).
  - **2-Visibility**
  - **3-Relationships**



OTHERS IMPORTANT NOTES

Abstract Class

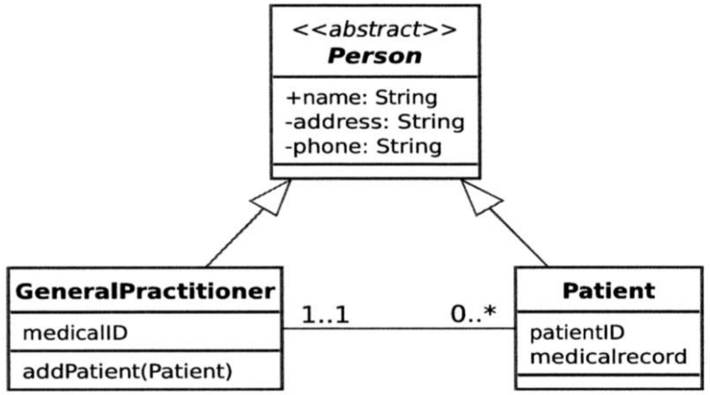
- Use **italic font** for the class name and abstract methods.
- Optionally label with «abstract».

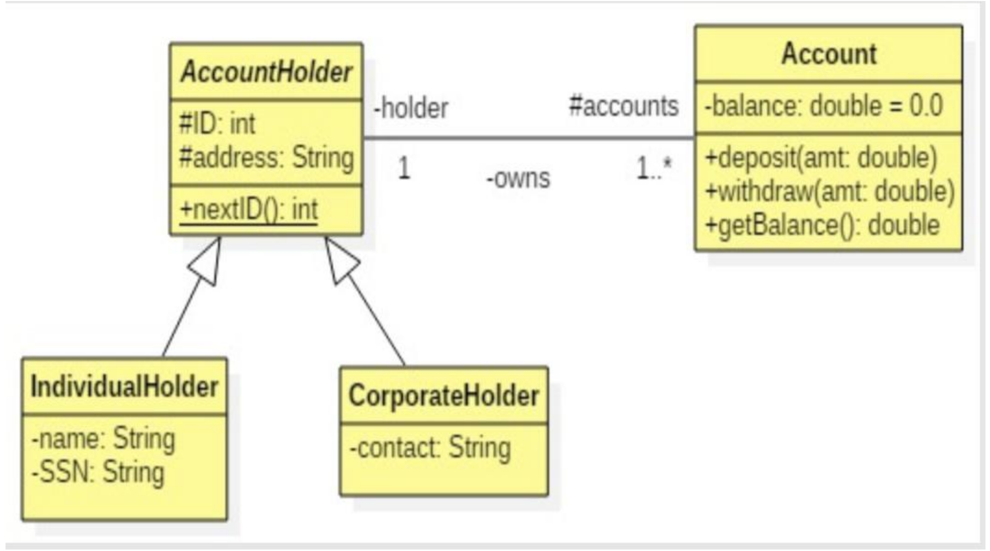
Interface

- Use «interface» or <<interface>> stereotype above the class name.
- All methods are **implicitly public and abstract**.
- Class that implements an interface uses **dashed arrow with empty triangle**.

Static

- **static members** (fields or methods) are **underlined**





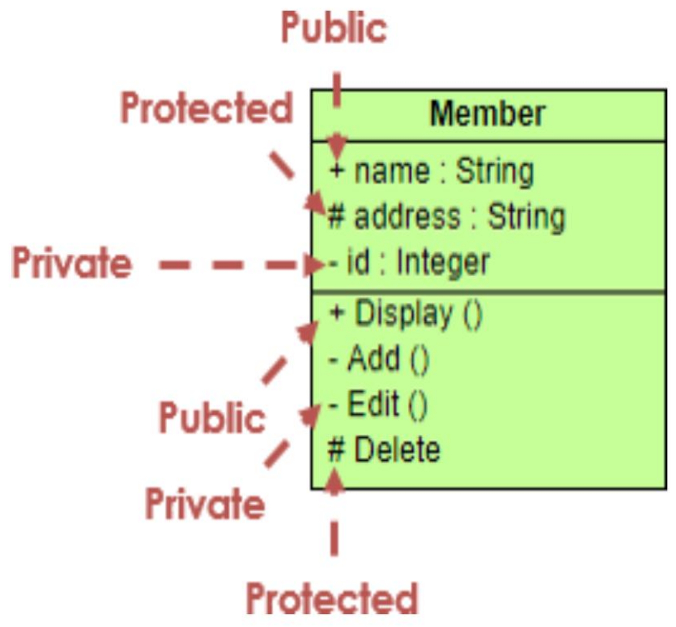
VISIBILITY

1-public element is visible to all elements that can access the contents of the namespace that owns it. Public visibility is represented by '+' literal.

2-protected element is visible to elements that have a generalization relationship to the namespace that owns it. Protected visibility is represented by '#' literal.

3-private element is only visible inside the namespace that owns it. Private visibility is represented by '-' literal.

4-package element is owned by a namespace that is not a package, and is visible to elements that are in the same package as its owning namespace. Only named elements that are not owned by packages can be marked as having package visibility. Any element marked as having package visibility is visible to all elements within the nearest enclosing package (given that other owning elements have proper visibility). Outside the nearest enclosing package, an element marked as having package visibility is not visible. Package visibility is represented by '~' literal.

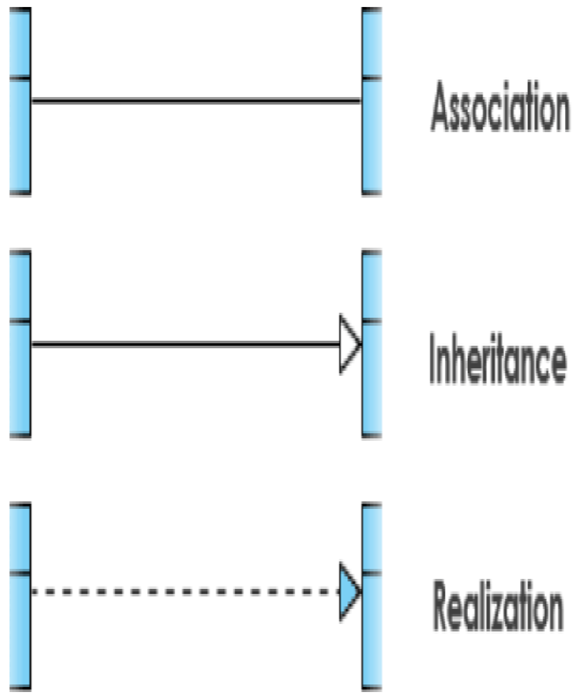


RELATIONSHIPS

1-Association:
This relationship refers to the collaboration and connection between two classes (the relationship between a teacher and a student).

2-Inheritance:
In this relationship, the class at the beginning of the arrow inherits from the class at the end (the relationship between a dog and an animal)

3-Realization:
This relationship is used to represent an interface and its implementing class. The class at the end of the arrow represents the interface that must be implemented in the classes on the other side of the relationship (the relationship between a Site Search and a Search Service interface).



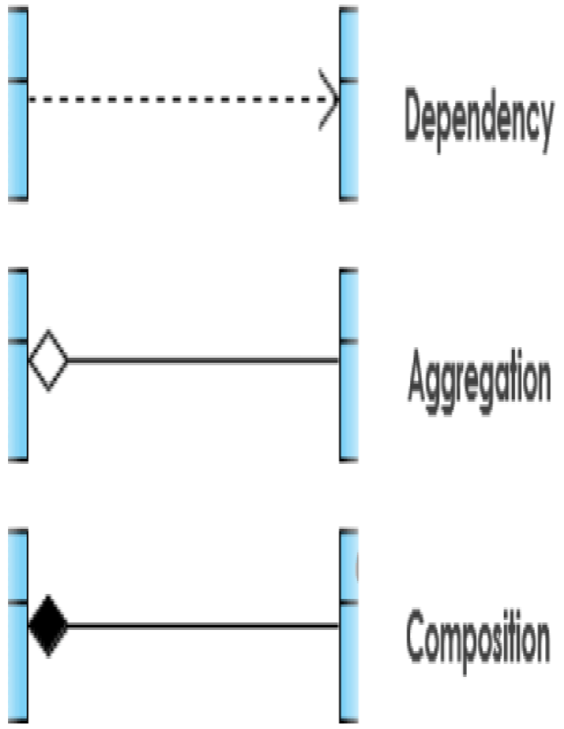
RELATIONSHIPS 2 *

* For those interested

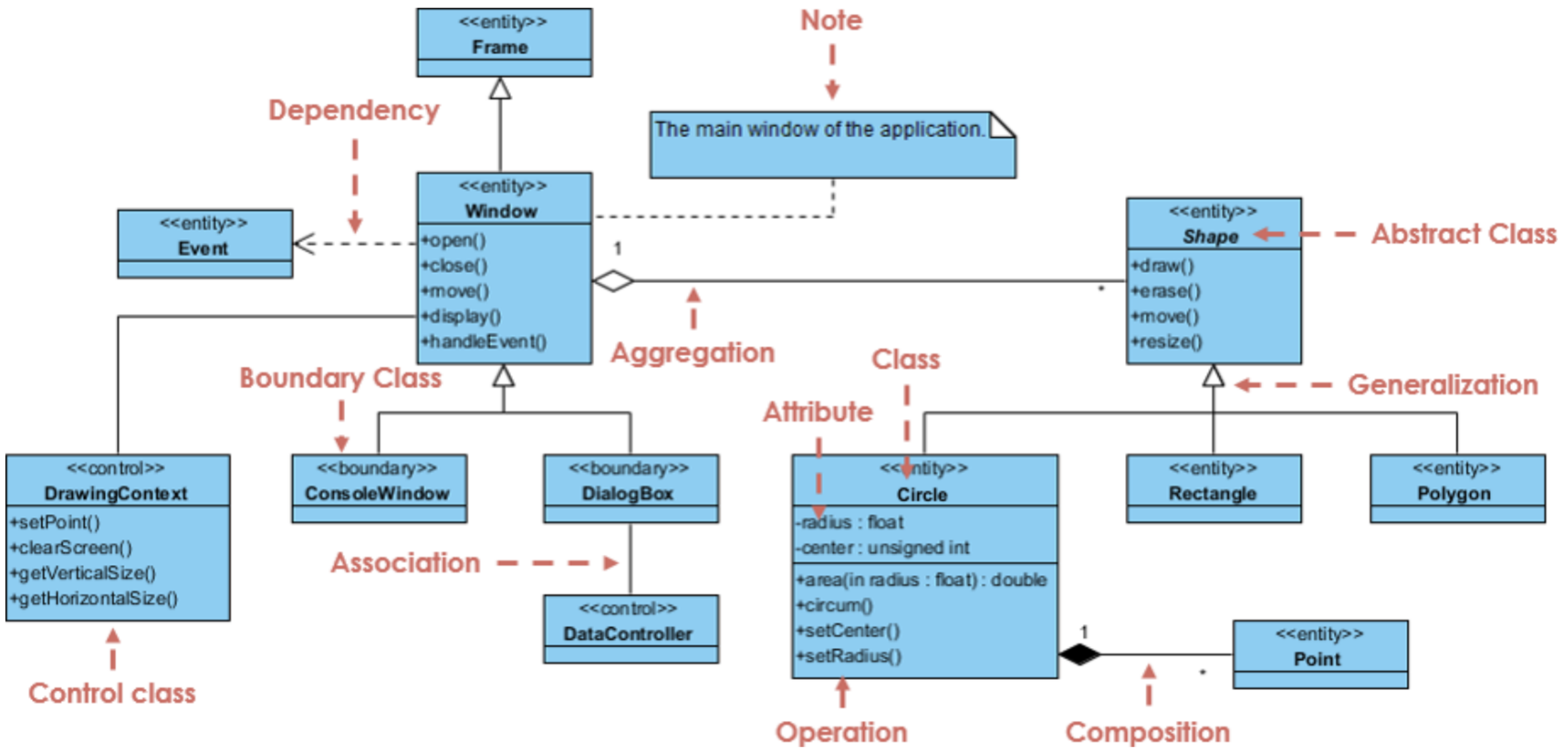
Dependency: In this relationship, the class at the beginning of the arrow requires the class at the end of the arrow to function properly (the relationship between a customer and a supplier).

Aggregation: In this relationship, the objects of the class on one side of the diamond have components that are made up of objects of the class on the other side. The components have a meaningful existence without the whole (the relationship between a teacher and a faculty).

Composition: The whole has a part relationship, but the existence of the parts without the whole is meaningless (the relationship between a room and a building).

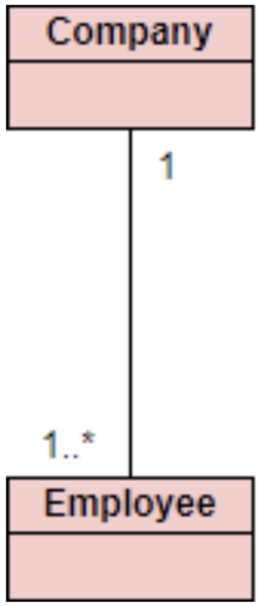


EXAMPLE OF RELATIONSHIPS



MULTIPLICITY *

* For those interested



Multiplicity	Option	Cardinality
0..1		No instances or one instance
1..1	**1**	Exactly one instance
0..*	*****	Zero or more instances
1..*		At least one instance
5..5	**5**	Exactly 5 instances
m..n		At least m but no more than n instances

2022-FINAL CLASS DIAGRAM DESIGN

- در این سوال قرار است که برای یک سری از فعالیتهایی که در **دانشگاه** انجام میشود طراحی انجام داده و نمودار کلاس آن را بکشید در دانشگاه دو گروه از افراد حضور دارند گروه اول دانشجویان هستند که به دو دسته دانشجویان لیسانس و تحصیلات تکمیلی تقسیم میشوند گروه دوم کارمندان هستند که به دو دسته کارکنان آموزش و اعضای هیأت علمی تقسیم میشوند. هر فرد یک شناسه نام و نام خانوادگی جنسیت و سال تولد دارد برای اعضای هیأت علمی چهار سطح تعریف میشود: مربی استادیار دانشیار و استاد تمام دانشجویان تحصیلات تکمیلی نیز میتوانند ارشد و یا دکترا باشند.
- سه نوع درس در دانشگاه ارائه میشود در اینجا برای سادگی لیست دروس مختصر شده است. نوع اول دروس عادی است شامل ریاضی و برنامه نویسی پیشرفته نوع دوم کارگاهها هستند شامل کارگاه برنامه نویسی و کارگاه عمومی آخرین نوع هم دروس تربیت بدنی هستند شامل فوتبال و شنا هر درس یک شناسه و تعداد واحد مشخصی دارد دروس عادی قابلیت حذف دارند ولی دروس کارگاه و تربیت بدنی امکان حذف ندارند برای دروس تربیت بدنی جنسیت دانشجویان مهم است.
- در هر ترم تعدادی کلاس همینهایی که هر ترم میگیرید در دانشگاه ارائه میشود هر کلاس برای یک درس تعریف شده است و یک شناسه ظرفیت و محل برگزاری دارد که به صورت رشته ذخیره میشود برای هر کلاس نیاز است که لیست دانشجویان ثبت نام کننده و مدرس آن را نیز نگهداری کنیم مدرس میتواند هیأت علمی و یا دانشجوی تحصیلات تکمیلی باشد. هر دانشجو میتواند انتخاب واحد کند و یا یک درس را حذف کند مدرس درس میتواند لیست دانشجویان یک کلاس را مشاهده کرده و یا نمره آنها را در سامانه ثبت کند.
- کارکنان آموزش میتوانند لیست تمام دانشجویان را مشاهده و اطلاعات کامل هر دانشجویی را مشاهده کنند همچنین میتوانند یک دانشجو را از لیست یک کلاس حذف و یا به لیست اضافه کنند یک درس را برای دانشجو بگیرند اساتید علاوه بر تدریس میتوانند دانشجویان را راهنمایی کنند.

2024-MIDTERM CLASS DIAGRAM DESIGN

در این مسئله قصد داریم سامانه‌ای برای مدیریت یک دانشگاه طراحی کنیم. افراد موجود در دانشگاه در این دانشگاه دو دسته اصلی از افراد حضور دارند: دانشجویان که شامل دو گروه هستند: دانشجویان کارشناسی و دانشجویان تحصیلات تکمیلی (کارشناسی ارشد و دکترا) و کارکنان که شامل دو گروه هستند: کارکنان اداری که ممکن است قراردادی یا رسمی باشند، و اعضای هیئت علمی که در یکی از سطوح استادیار، دانشیار یا استاد تمام قرار دارند.

- اطلاعات پایه افراد

هر فرد در دانشگاه (چه دانشجو و چه کارمند) دارای ویژگی‌های پایه‌ای زیر است: کد شناسایی، نام، نام خانوادگی، جنسیت و سال تولد. کارمندان یکسری اطلاعات اضافه تر شامل مدت قرارداد و ... دارند

- دانشگاه

دانشگاه باید دارای اطلاعات پایه مانند نام، نشانی، بودجه کل و فهرستی از تمامی افراد (دانشجویان و کارکنان) باشد.

- فرم رفاهی

سامانه دارای بخشی برای خدمات مالی و رفاهی است و اطلاعات مالی مانند بودجه باقی‌مانده برای وام و فهرست وام‌های ثبت‌شده را نگهداری می‌کند.

- وام

وام نیز در سامانه وجود دارد و شامل دو نوع وام است: وام بدون بهره و وام دارای بهره.

هر وام دارای ویژگی‌هایی مانند مبلغ، مدت بازپرداخت، نرخ بهره و ... می‌باشد. نرخ بهره باید به‌صورت یک نوع شمارشی شود، مثلاً شامل سطوح کم، متوسط و زیاد.

INTRODUCTION TO SQLITE - CRUD

Definition: Lightweight, file-based relational database system.

Key Features:

- **Serverless**, zero-configuration, single database file.
- ACID-compliant transaction

Comparison:

- **SQLite:** Simple, no server overhead, great for small/medium apps.
- **MySQL/PostgreSQL/Oracle:** Requires server process, more complex setup, handles larger concurrent loads.

When to Use SQLite:

- Prototyping, local data storage, low concurrency needs.

TABLE

```
-- Parent table
CREATE TABLE departments (
    dept_id INTEGER PRIMARY KEY
     AUTOINCREMENT,
    name TEXT NOT NULL,
    location TEXT
);
```

```
-- Child table with foreign key
CREATE TABLE employees (
    emp_id INTEGER PRIMARY KEY
     AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE,
    salary REAL,
    dept_id INTEGER,
    FOREIGN KEY (dept_id) REFERENCES
     departments(dept_id) ON DELETE CASCADE
);
```

CREATE

```
-- Insert into parent table first
INSERT INTO departments (name, location) VALUES ('Engineering', 'Floor 5');
INSERT INTO departments (name, location) VALUES ('Marketing', 'Floor 2');

-- Insert into child table with valid foreign key
INSERT INTO employees (name, email, salary, dept_id)
VALUES ('John Doe', 'john@example.com', 75000, 1);

-- This will fail due to foreign key constraint (no department with id 99)
INSERT INTO employees (name, email, salary, dept_id)
VALUES ('Invalid', 'invalid@example.com', 50000, 99);
```

READ

```
-- Simple SELECT
SELECT * FROM employees;
-- Filtering
SELECT * FROM employees WHERE salary > 60000;
-- Sorting
SELECT * FROM employees ORDER BY name ASC;

-- Get employees with their department names
SELECT e.first_name, e.last_name, d.dept_name
FROM employees e, departments d
WHERE e.dept_id = d.dept_id;
```

UPDATE

```
-- Update employee salary
UPDATE employees SET salary = 80000 WHERE emp_id = 1;

-- Update department (foreign key must exist)
UPDATE employees SET dept_id = 2 WHERE emp_id = 1;

-- This will fail if department 99 doesn't exist
UPDATE employees SET dept_id = 99 WHERE emp_id = 1;
```

DELETE

```
-- Delete an employee
DELETE FROM employees WHERE emp_id = 1;

-- Delete a department (will cascade to delete all its employees)
DELETE FROM departments WHERE dept_id = 1;
```

JDBC DRIVER WITH MAVEN

```
import java.sql.*;


public class SQLiteMinimalCRUD {
    public static void main(String[] args) {
        try (Connection conn =
     DriverManager.getConnection("jdbc:sqlite:sample.db");
             Statement stmt = conn.createStatement()) {

            ResultSet rs = stmt.executeQuery("SELECT *
     FROM products");
while (rs.next()) {
                System.out.printf("%d: %s - $%.2f (Stock:
    %d)%n",
                        rs.getInt("id"),
                        rs.getString("name"),
                        rs.getDouble("price"),
                        rs.getInt("stock"));
            }
        } catch (SQLException e) {
            System.err.println("Database error: " +
    e.getMessage());
        }
    }
```

پایان