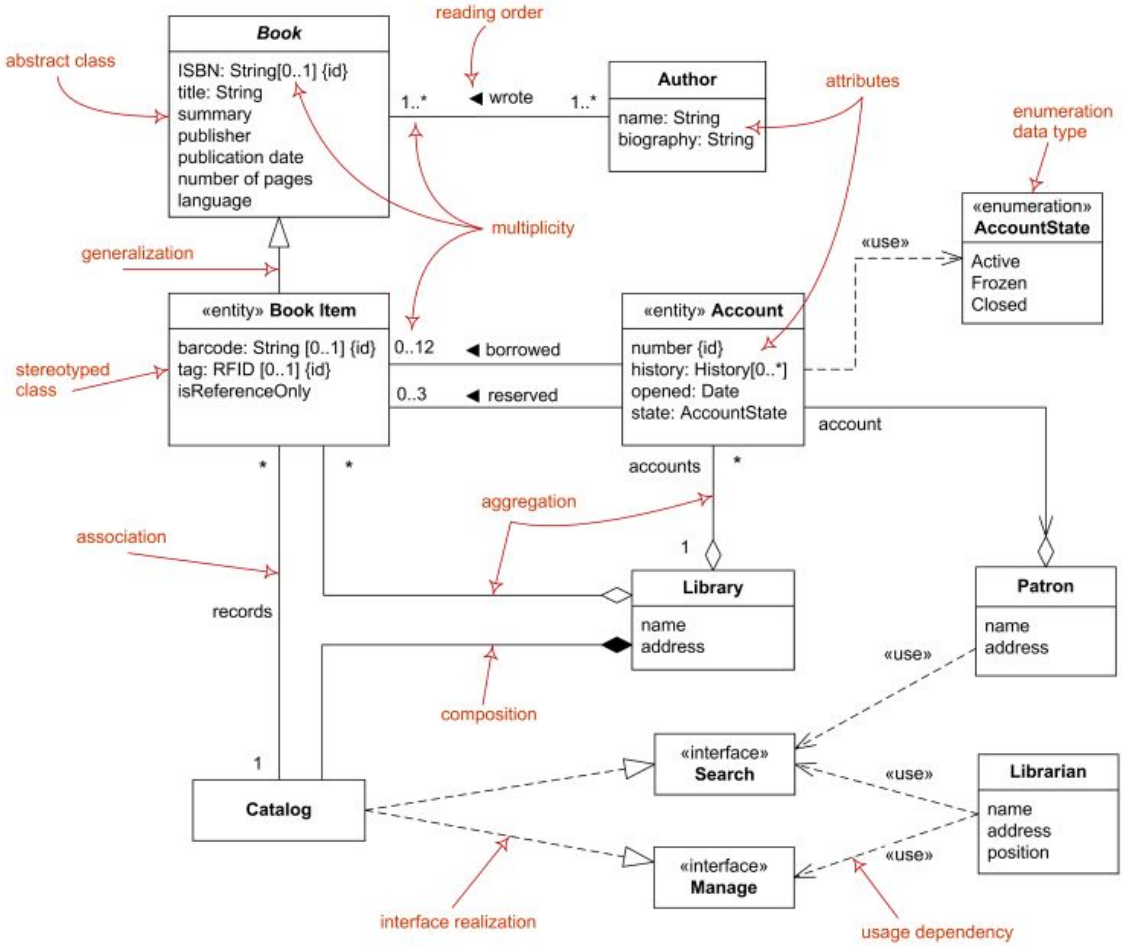
Low Level Design (LLD)

Part 1

What is LLD?

- Component Level Design Process
- Provides internal logic of software being developed
- Defines the class diagram that has attributes and methods, defining relationships between them, etc.


Book
ISBN: String[0..1] {id}
title: String
summary
publisher
publication date
number of pages
language
abstract class
reading order
1..*
◀ wrote
1..*
Author
name: String
biography: String
attributes
enumeration
data type
«enumeration»
AccountState
Active
Frozen
Closed
«use»
multiplicity
generalization
«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
isReferenceOnly
stereotyped
class
0..12
◀ borrowed
0..3
◀ reserved
«entity» Account
number {id}
history: History[0..*]
opened: Date
state: AccountState
account
accounts
*
*
*
aggregation
1
association
Library
name
address
Patron
name
address
records
«use»
composition
1
«interface»
Search
«use»
Librarian
name
address
position
Catalog
«interface»
Manage
«use»
interface realization
usage dependency

Fundamentals of LLD

OOP

Object-Oriented Programming

Course Structure

Week 1	Foundational OOP
Week 2	Practical OOP
Week 3	Advance Concepts of OOP
Week 4	LLD Basic Principles
Week 5	Design Patterns
Week 6	How to approach LLD problems?
Week 7	LLD FAQs
Week 8	LLD FAQs

What is OOP?

Object-Oriented Programming (OOP) is a fundamental concept in software development that revolves around the concept of **Classes and Objects**.

Classes and Objects

```
int main() {
    int a;
    ....
    return 0;
}
```

Classes and Objects

```
int main() {
    int a;
    ....
    return 0;
}
```

32-bit integer value

Classes and Objects

```
class Person {
public:
    int age;
    string name;
    string address;

};

int main() {
    int a;
    Person p1, p2;
    ....
    return 0;

}
```

Classes and Objects

```
class Person {
public:
    int age;
    string name;
    string address;

};
```

Custom defined.

```
int main() {
    int a;
    Person p1, p2;
    ....
    return 0;

}
```

Classes and Objects

```
class Person {
public:
    int age;
    string name;
    string address;

};
```

Custom defined.

```
int main() {
    int a;
    Person p1, p2;
    ....
    return 0;

}
```

Non-primitive data-type

Classes and Objects

```
class Person {
public:
    int age;
    string name;
    string address;

};
```

Custom defined.

```
int main() {
    int a;
    Person p1, p2;
    ....
    return 0;

}
```

Non-primitive data-type

Objects

Classes and Objects

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();
    return 0;
}
```

Classes and Objects

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();
    return 0;
}
```

Class members are accessed by (.) operator

Classes and Objects

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();

    p1.age_ = 25;

    return 0;
}
```

Classes and Objects

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();

    p1.age_ = 25;

    return 0;
}
```

Won't compile

Classes and Objects

The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();

    p1.age_ = 25;

    return 0;
}
```

Won't compile

Classes and Objects

The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

```
class Person {
public:
    int GetName() {
        return name;
    }
    void SetName(string name) {
        name_ = name;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

can be accessed here

```
int main() {
    Person p1;
    p1.SetName("abc");
    cout << p1.GetName();

    p1.age_ = 25;

    return 0;
}
```

Won't compile

The private members are not accessible outside the class; they can be accessed only through member functions of the class.

Class Initialization

Class Initialization

Constructors

A constructor is a special member function that is automatically called after a object is created.

Constructor

```
class Person {
public:
    Person(int age, string name, string address) {
        age_ = age;
        name_ = name;
        address_ = address;
    }
    ...

private:
    int age_;
    string name_;
    string address_;

};
```

Class Initialization

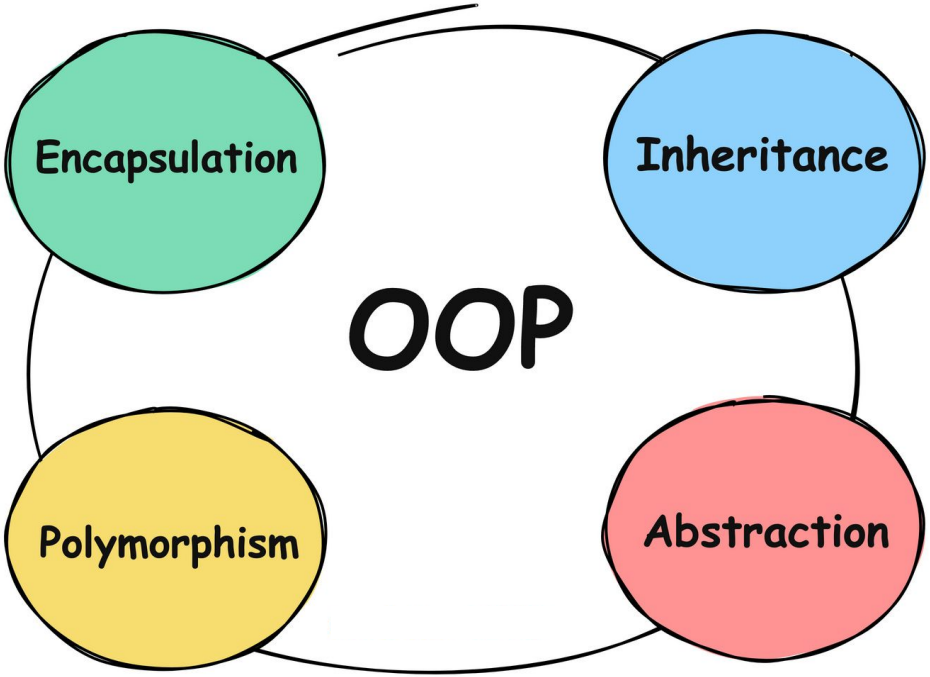
Constructors

Unlike normal member functions, constructors have specific rules for how they must be named:

- Constructors must have the same name as the class (with the same capitalization).
- Constructors have no return type **(not even void)**.

Concepts Check-in

- LLD Overview
- Classes and Objects
- Public & Private members
- Constructors
- Getter & Setter Pattern

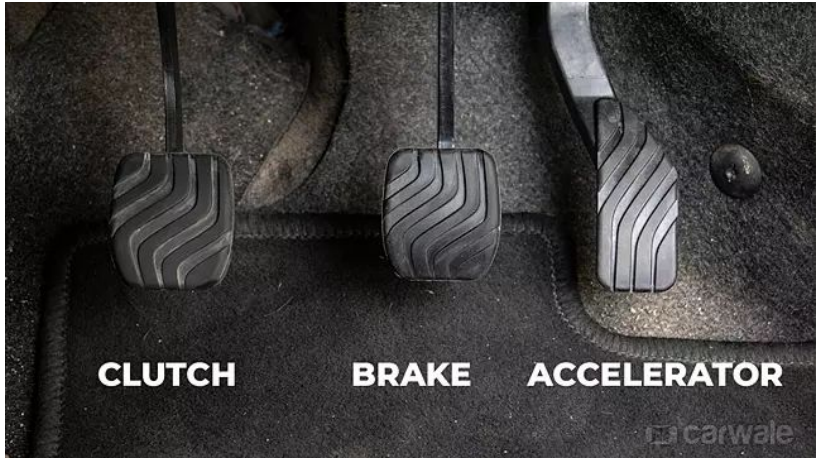

Encapsulation
Inheritance
OOP
Polymorphism
Abstraction

Encapsulation

- Encapsulation is the integration of data and operations into a class.
- Encapsulation is hiding the functional details from the object calling it.

Encapsulation

- Encapsulation is the integration of data and operations into a class.
- Encapsulation is hiding the functional details from the object calling it.



Encapsulation

```
class Car {
public:
    void Clutch();
    void Break();
    void Accelerate();

private:
    Engine engine_;
    ....
};
```

Exposed to the user

Hidden from the user

Inheritance

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object.

Inheritance represents IS-A relationship.

E.g. Employee is a Person, Engineer is a Employee, etc.

Inheritance

```
class Person {
public:
     int age;
     string name;
     string address;
     string mobile_no;
     string aadhaar_no;
};
```

```
class Employee : public Person {
public:
     string job_role;
     string company;
     int salary;
     int id;
};
```

```
class SoftwareEngineer : public Employee {
public:
     string team;
     string tech_stack;
     string language_expertise;
     int level;
};
```

Inheritance

```
class SoftwareEngineer {
public:
     string team;
     string tech_stack;
     string language_expertise;
     int level;

     string job_role;
     string company;
     int salary;
     int id;

     int age;
     string name;
     string address;
     string mobile_no;
     string aadhaar_no;
};
```

What’s next?

- Polymorphism
- Abstraction

Thank You!