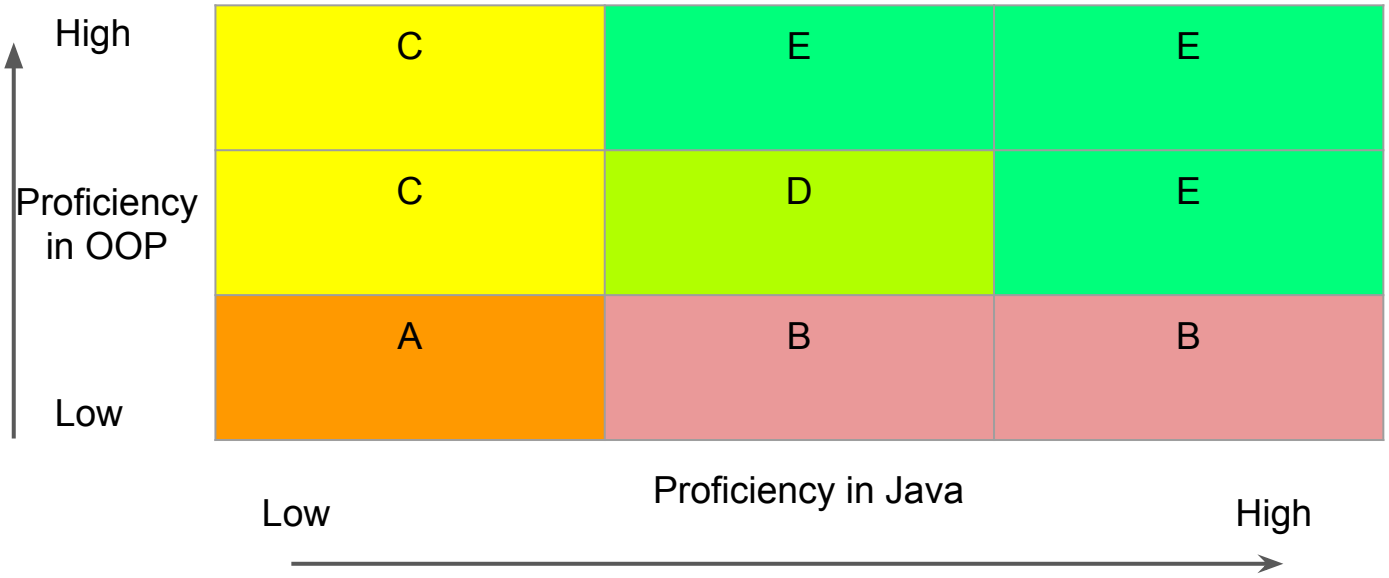
ESS 201: Programming in Java

Composition
Term 1: 2020-21
T K Srikanth
IIIT B

Java and OOP - where should you be?



Goals of Object-oriented design

Modularity

Abstractions

Re-use

Correctness

Robustness & Quality

Collaboration

Productivity

Maintainability

Importance of Design

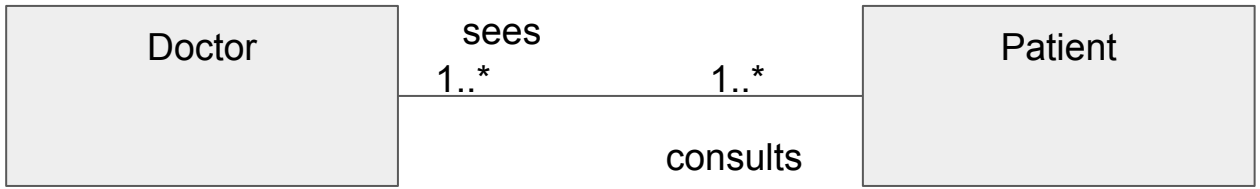
Give me six hours to chop down a tree and I'll spend the first four sharpening my axe

- Abraham Lincoln

Relationships between objects

Association: a weak relationship between objects. E.g. Doctor and Patient

- A path of communication or link between objects, where one object “uses” possibly multiple instances of the other. No ownership or other semantics are implied.
- The lifecycles of the objects are independently defined



Association

Doctor <--> Patient

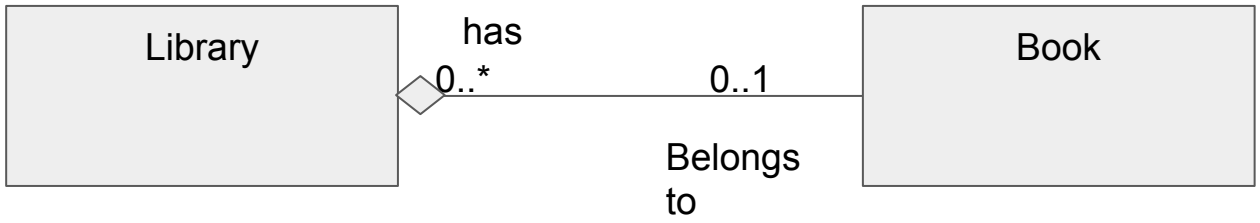
Book <--> Person (reader)

Person <--> Person (friend)

Relationships: Aggregation

An object consists of (or is made up of) instances of objects of another class, but the lifecycle of the two objects are independent. There is a “has a” relationship between the first object and the second.

E.g. Library and Books



Aggregation

Room --- Furniture

Bus --- Passenger

Course --- Student

Computer --- Battery, Display, Mouse, …

Team --- Engineer

Relationships: Composition

An object contains (and *owns*) instances of another object, and the second object does not exist if the owning object is deleted.

E.g. Bank and Account

Existence dependency: Component exists only when its container is around

College --- Course

College --- Department

Company --- Team

Person --- Hand/Leg/Head

Composition - Re-use of Implementation

Composition enables the design of complex classes by re-using existing classes: we re-use implementation of these classes

The new class is able to leverage all the functionality of one or more existing classes.

However, it typically hides the existence of these classes - hopefully keeping the references private, and exposes new interfaces.

Inheritance: Re-use of behaviour and state

Derive a new class by ***extend***-ing an existing class

Case 1: Reuse all (non-private) methods of a class and **add** new behaviour/state. Existing class used “as is”

Case 2: **Modify** one or more methods of a class to change the functionality of the derived class. **Override** behaviour

Case 3: **Implement** one or more methods of the base class, where the base class defines only the method interface but not the implementation. Concrete implementation of an **abstract** method

A given “**extends”** may involve one or more of these possibilities.

Java supports only “single inheritance”- can “*extend*” only one class