Software Re-Engineering

Lecture: 06

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer &
Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- How to Develop a Domain Model
- Features of Domain Model
- Examples of Domain Model
- Case Study: Local Hospital Problem

Domain Model

- Shows the real-world flow, relationship between entities and concepts (e.g. entities can be objects and concepts may contain objects with different attributes)
- Captures the most important types of objects in a system.
- Describing “things” in a system and how these things are related to each other.
- A “thing” can be an object, a class, an interface, a package, component or a subsystem, which is part of the system being developed.
- Very important process because it is employed throughout the entire system development life cycle.

Characteristics of Domain Model

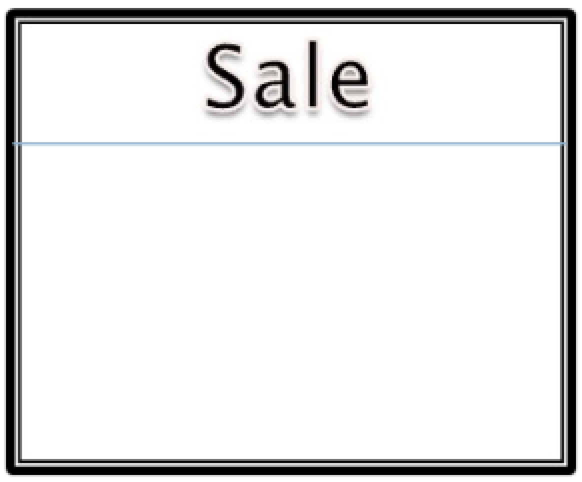
- Visual representation of conceptual classes.
- Associations and relationships between concepts (e.g Payment PAYS-FOR Sales). Attributes for information content (e.g. Sale records DATE and TIME).
- Does not include operations/ functions.
- Does not describe software classes.
- Does not describe software responsibilities.

Features of Domain Model

- Domain Classes
- Entities
- Attributes
- Relationships
- Aggregate
- Composition

Domain Classes

- Each domain class denotes a type of object.



Entities

- Entities are the fundamental building blocks of a domain model

- In most cases, they are nouns distinguished by the attributes they have.

- For instance, in a banking space, the entities could relate to Customer, Account, Transaction, etc.

Attributes

- Attributes are characteristics that describe entities.
- They provide additional information about entities and contribute to their definition.
- For instance Customer entity may include the attributes such as the name, address, phone number, and so on

Relationships

- Entities rarely exist in isolation; they are connected through relationships.

- The relationships define how entities interact with each other.

- Different types of relationships generally fall into one-to-one, one-to-many, or many-to-many categories. Aggregates and Composition: In more complex scenarios, groups of related entities can be combined into aggregates.

Aggregate

- Implies a relationship where the child can exist independently of the parent.

- Example: In OOP, Main Class: Class (parent) and, Derived Class: Student (child). Delete the Class and the Students still exist.

Composition

- Implies a relationship where the child cannot exist independent of the parent.
- Example: Main Class: House (parent) and, Derived Class Room (child). Rooms don't exist separate to a House

Actions and Behaviors

- Entities within a domain don't just exist; they engage in actions and behaviors.
- These actions can be modeled through methods or operations that entities can perform.
- Behaviors capture the functionality and business logic associated with entities.
- For example, a "Customer" entity might have a "PlaceOrder" behavior.

Inheritance and Generalization

- Inheritance allows entities to inherit properties and behaviors from other entities, enabling the creation of specialized entities that share common characteristics.

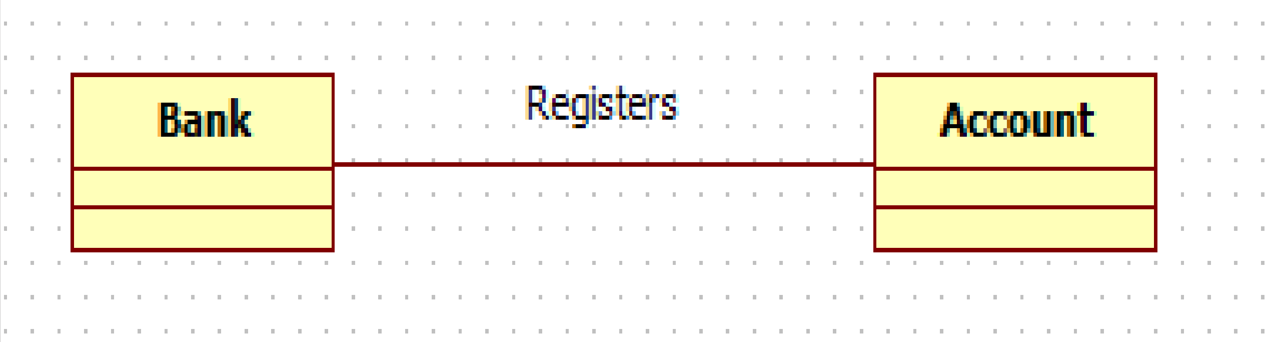
- This relationship aids in modeling hierarchies and categories within the domain.
- An example could be a "PremiumCustomer" entity inheriting from the "Customer" entity.

Steps to Create Domain Model

- Create User Stories
- Identify candidate conceptual classes
- Draw them in a UML domain model
- Add associations necessary to record the relationships
- Add attributes necessary for information to be preserved
- Use existing names for things, the vocabulary of the domain

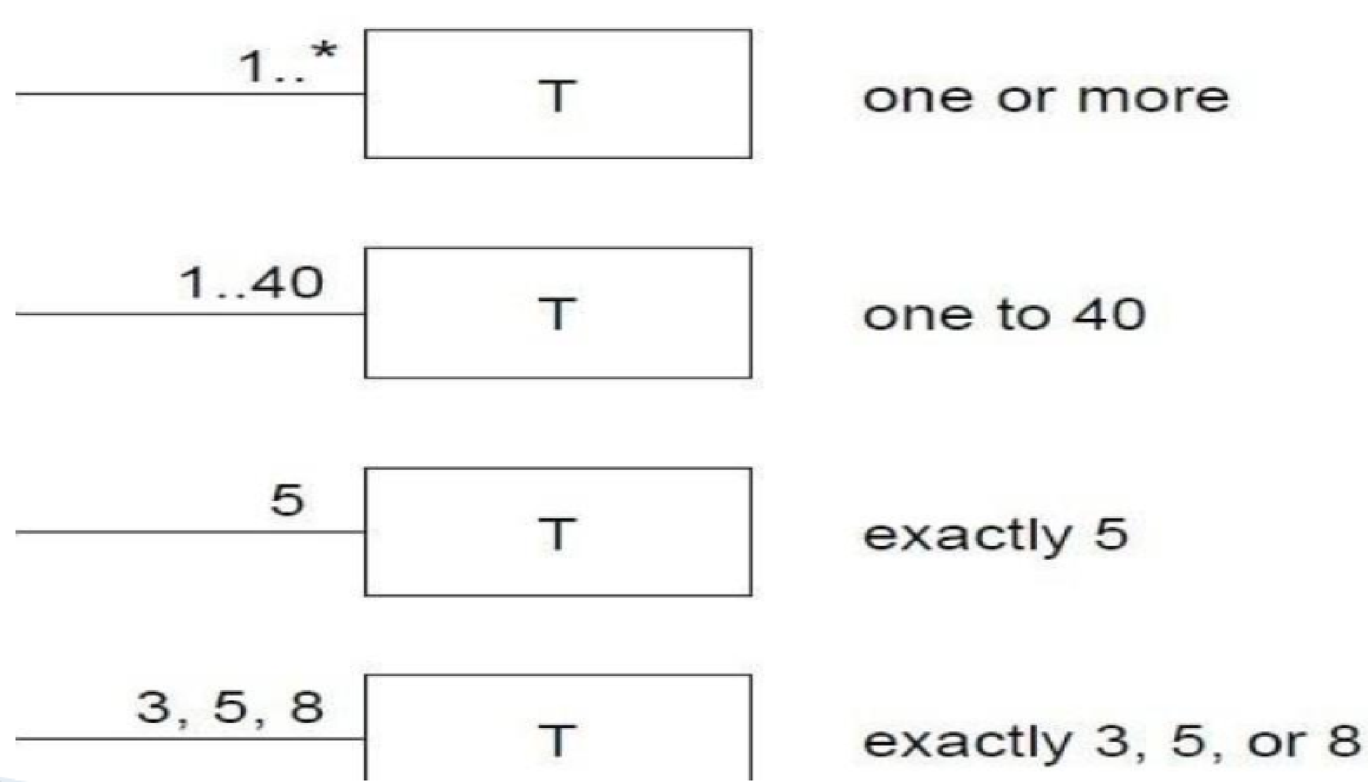
Association

- It’s a relationship between two objects that defines multiplicity between objects that can be one-to-one, one-to-many, many-to-one, many-to-many all these concepts define an association between objects.
- For example: Students and Faculty are having an association)



Multiplicity

- Describes how many instances of one concept can be associated with one instance of the related concept

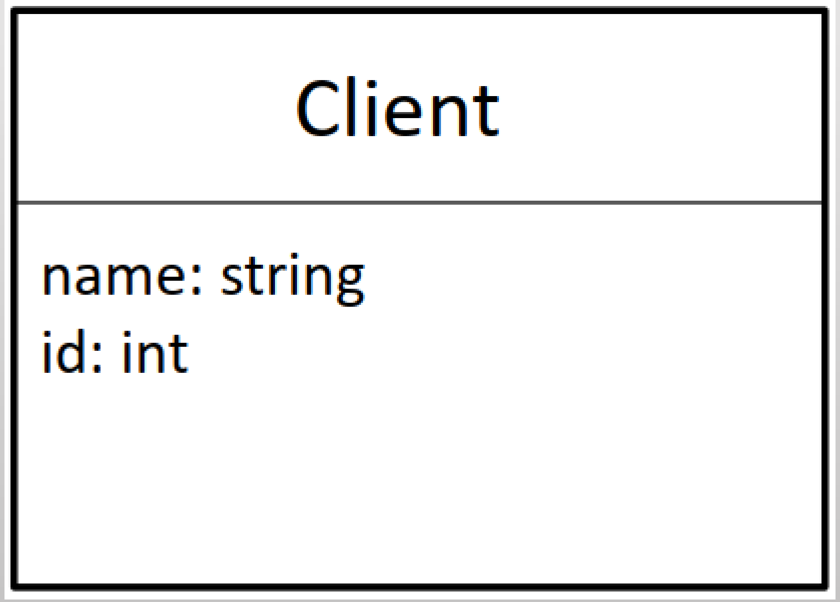


Role

- Each end of an association is called a role.
- Roles may have: Name, Multiplicity, Expression, or Navigability

Attributes

- Attributes refer to properties that define the class.
- For example: A class Client will have attributes name and id.



Client

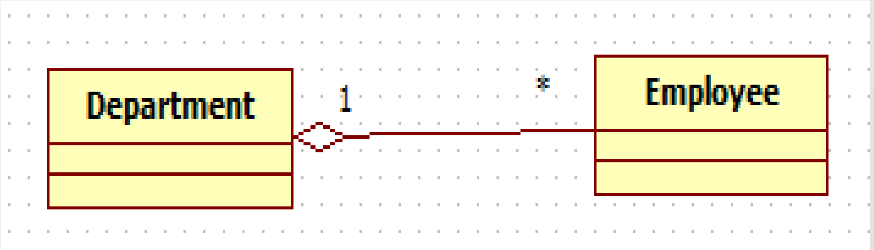
name: string
id: int

Class Visibility

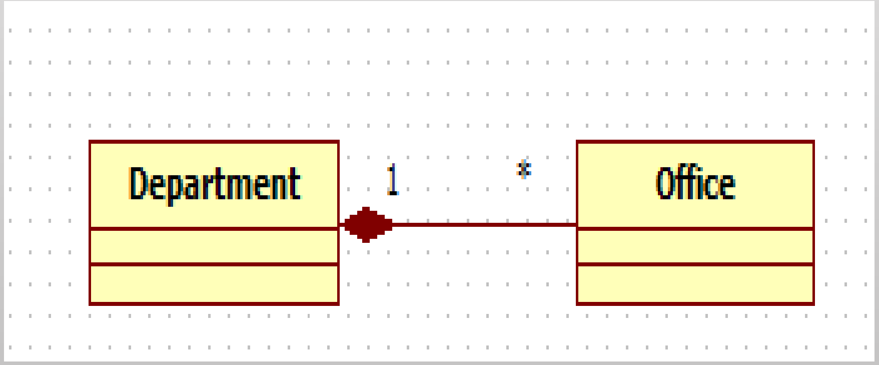
- The +, #, -, ~, and / symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + Denotes the public attributes or operations
- # Denotes protected attributes or operations
- - Denotes private attributes or operations
- ~ Denotes a package (a class, method or variable)
- / Denotes derived attribute

- Aggregation

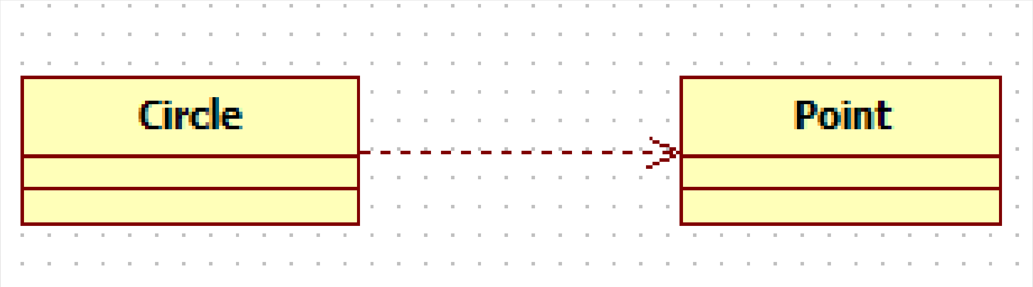


- Composition



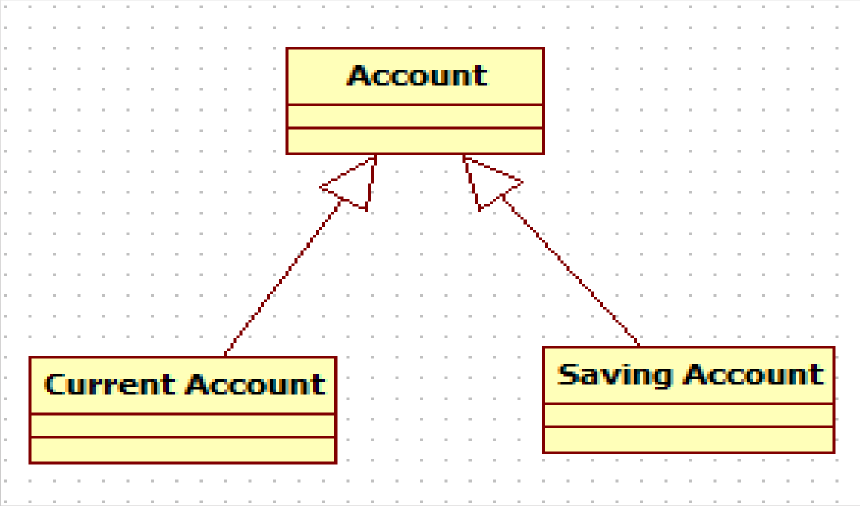
Dependency

- Change in the structure or behavior of a class affects the other related class, then there is a dependency between those two classes. (e.g. Relationship between points and circle)

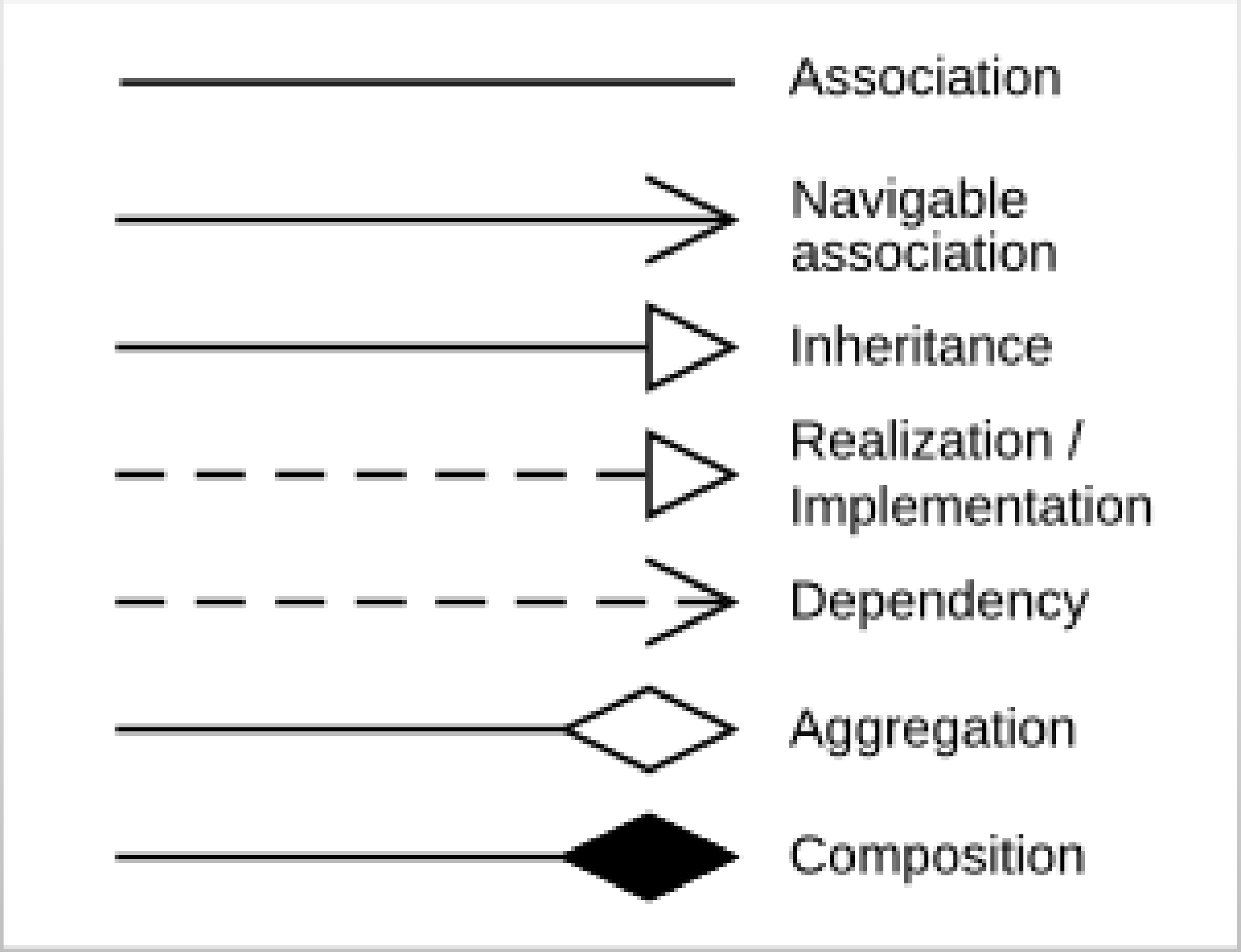


Generalization

- Generalization uses a “is-a” relationship from a specialization to the generalization class.



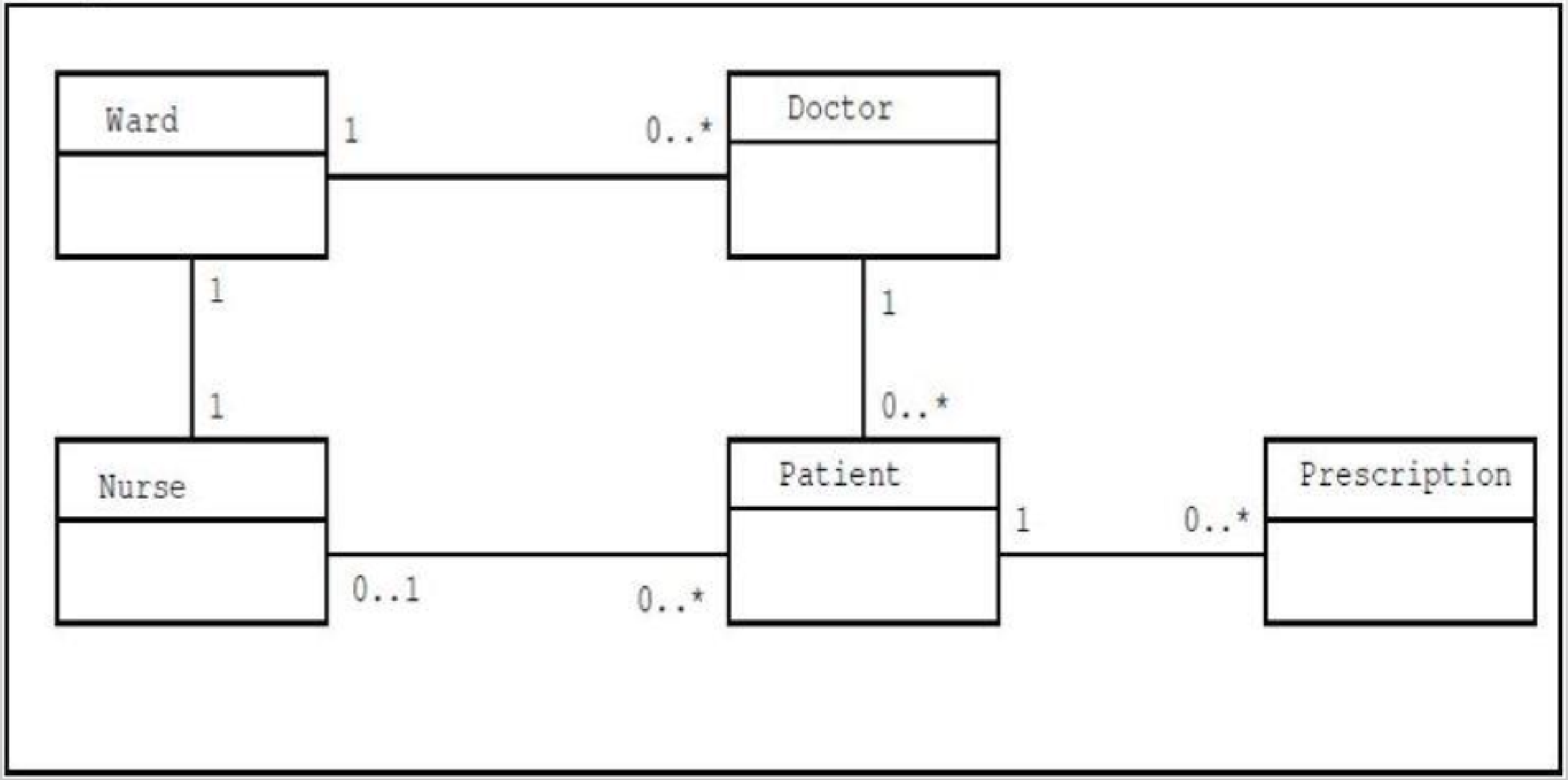
Symbols of Relationships



Case Study

- A Local Hospital consists of many wards, each of which is assigned many patients.
- Each **patient** is assigned to **one doctor**, who has overall responsibility for the patients in his or her care.
- Each **patient** is **prescribed drugs** by the **doctor** responsible for that patient.
- Each **nurse** is assigned to a **ward** and nurses **all the patients** in the **ward**.
- Each **patient** is assigned **one nurse** in this position of responsibility.
- Prospective **ward** is look-after by the prospective **doctors**.

Case Study



Thank You!