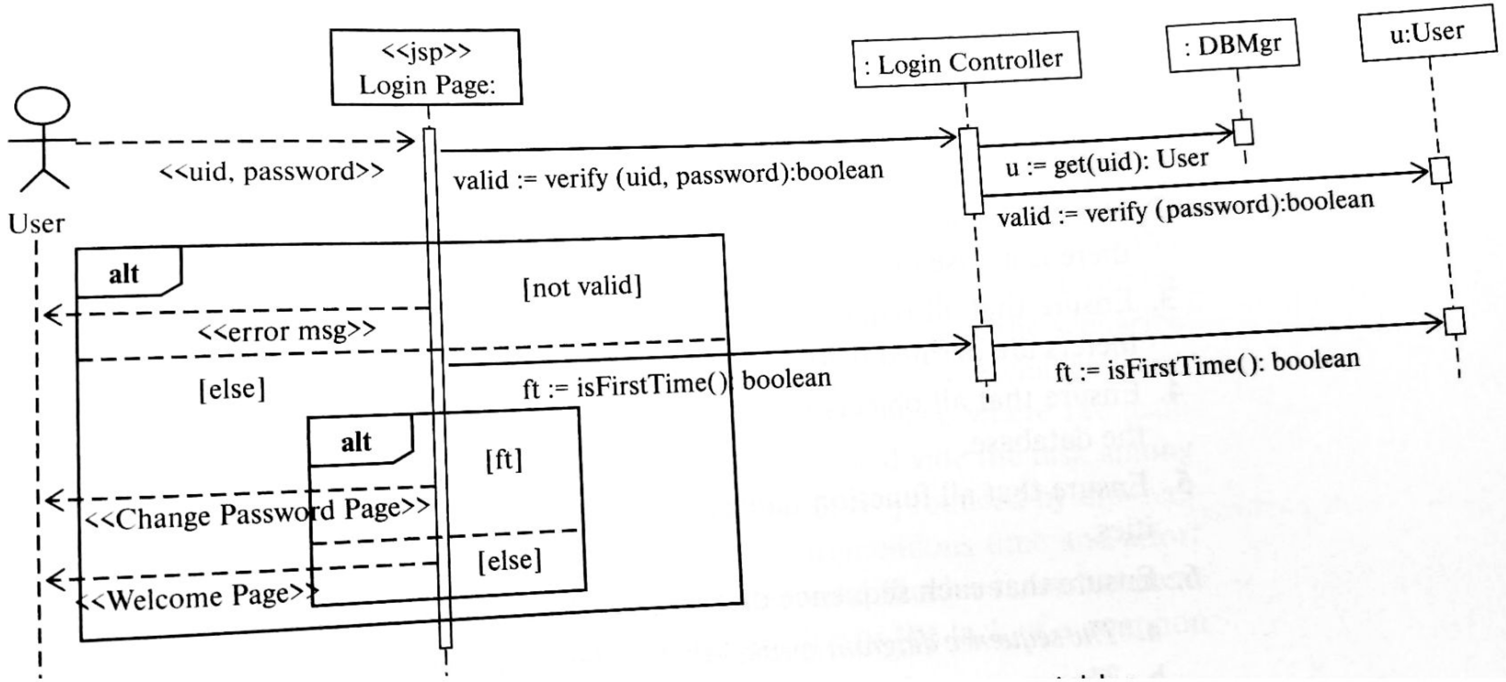
Inheritance vs Composition

Manas Jyoti Das, PhD
Computer Science


User
<<jsp>>
Login Page:
: Login Controller
: DBMgr
u:User
<<uid, password>>
valid := verify (uid, password):boolean
u := get(uid): User
valid := verify (password):boolean
alt
[not valid]
<<error msg>>
[else]
ft := isFirstTime(): boolean
ft := isFirstTime(): boolean
alt
[ft]
<<Change Password Page>>
[else]
<<Welcome Page>>

Inheritance

- Relationship: Represents an "is-a" relationship. A child class inherits properties and behaviors from a parent class, becoming a more specialized version of it
- Example: A class Car can inherit from a class Vehicle. Since a car is a type of vehicle, it inherits the general properties (e.g., number of wheels) and behaviors (e.g., move()) from the Vehicle class, while adding car-specific features (e.g., doors, engine type)
- Benefits:
    - Promotes code reuse by avoiding redundant code implementation in child classes.
    - Useful for modeling real-world hierarchical relationships, like animals inheriting from a common ancestor class

Inheritance

- Drawbacks:
- Tight coupling: Changes in the parent class can potentially impact all child classes, making maintenance more complex
- Limited flexibility: A child class can only inherit from one parent class (depends on programming language), restricting its potential behavior

Composition

- Relationship: Represents a "has-a" relationship. A class contains instances of other classes as its members, utilizing their functionalities to achieve its own purpose
- Example: A class Student can have-a class Address as a member. The Student class doesn't inherit from Address but utilizes the information and methods provided by the Address object to manage student address data
- Benefits:
  - Promotes loose coupling: Changes in one class have minimal impact on others, as they collaborate through defined methods.
  - More flexible: A class can have instances of multiple classes, allowing for more complex combinations of functionalities

Composition

- Drawbacks:
- Might require more code compared to inheritance for simpler scenarios.
- Doesn't establish an "is-a" relationship, which may not be suitable for all use cases

Examples of compositions

- House and Room:

- Class: House
- Member: List<Room> rooms
- Relationship: A House has-a list of Room objects. Each Room object can have its own properties and behaviors specific to different room types (bedroom, kitchen, etc.)
- Code..

Examples of compositions

Shopping Cart and Item:

- Class: ShoppingCart
- Member: List<Item> items
- Relationship: A ShoppingCart has-a list of Item objects. Each Item object represents a product with its details (name, price, quantity). The ShoppingCart class manages adding, removing, and calculating the total cost of items in the cart
- Code...

Which one to choose?

- The choice between inheritance and composition depends on the specific relationship you want to model
- Use inheritance when:
  - There's a clear "is-a" hierarchy between classes (e.g., animals inheriting from a mammal class).
  - You want to enforce code reuse and maintainability in a well-defined hierarchy.
- Use composition when:
  - You need a more flexible way to combine functionalities from different classes.
  - You want to avoid tight coupling between classes and maintain independence

Example

- Music player
- Think BIG!!!