ABAP OO
+ Test framework

João Vitor de Camargo, Suzana Machado, SAP
October, 2019

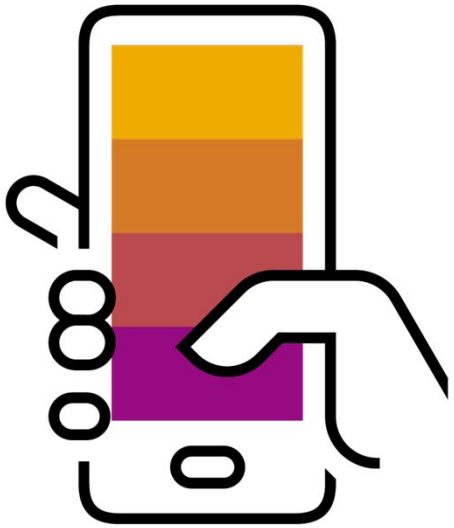
Day 1

- **Encapsulation**
- **Abstraction**
  - Interfaces
  - Abstract classes
- **Classes relationships**
  - Association
  - Aggregation
  - Composition
- **Inheritance**
- **Polymorphism**

Encapsulation

Encapsulation

- An object’s capacity of keeping its state private
- Information hiding
  - Hide internal representation, protect data from outside world;
- The class maintains its own state
  - Other classes only can access this state if explicitly allowed (via public methods);
  - Changes to the state of the object are made without minding the public interface ;
- Solves issues at **implementation** level
  - Focus on **how it should be done**;
- In other words, encapsulation means “hiding code and data into only one unit”



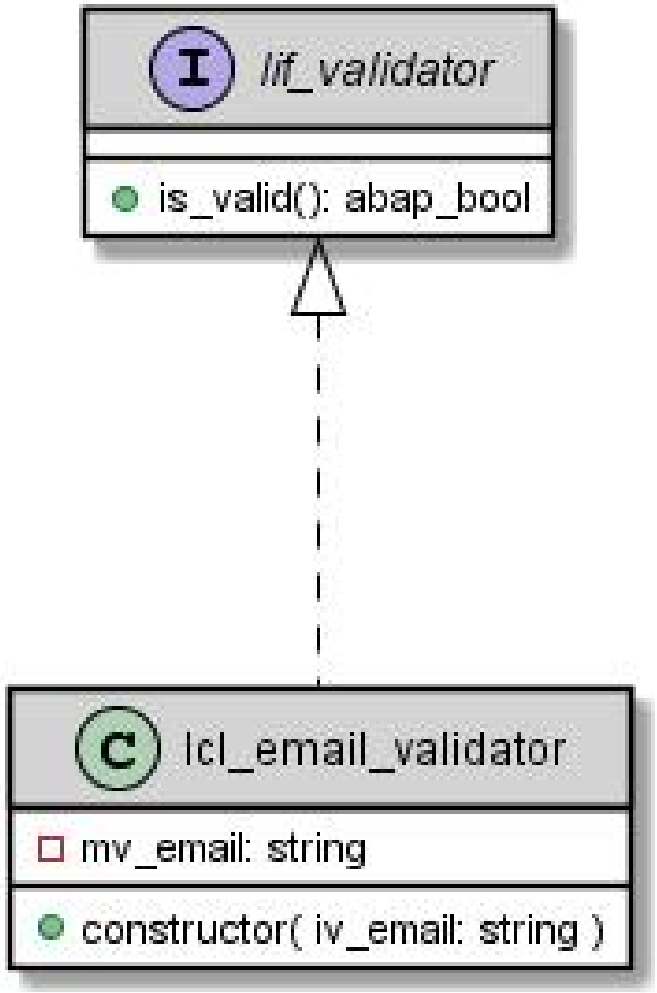
Abstraction

Abstraction

- Not associated with any particular instance
- Remove the concretion of a concept
  - Only the essential properties remain, to which we can apply some logic;
- Hides unnecessary details
  - Classes that use the abstraction only have to know what it does, not how it does;
- Solves issues at **design** level
  - Focus on **what should be done**;

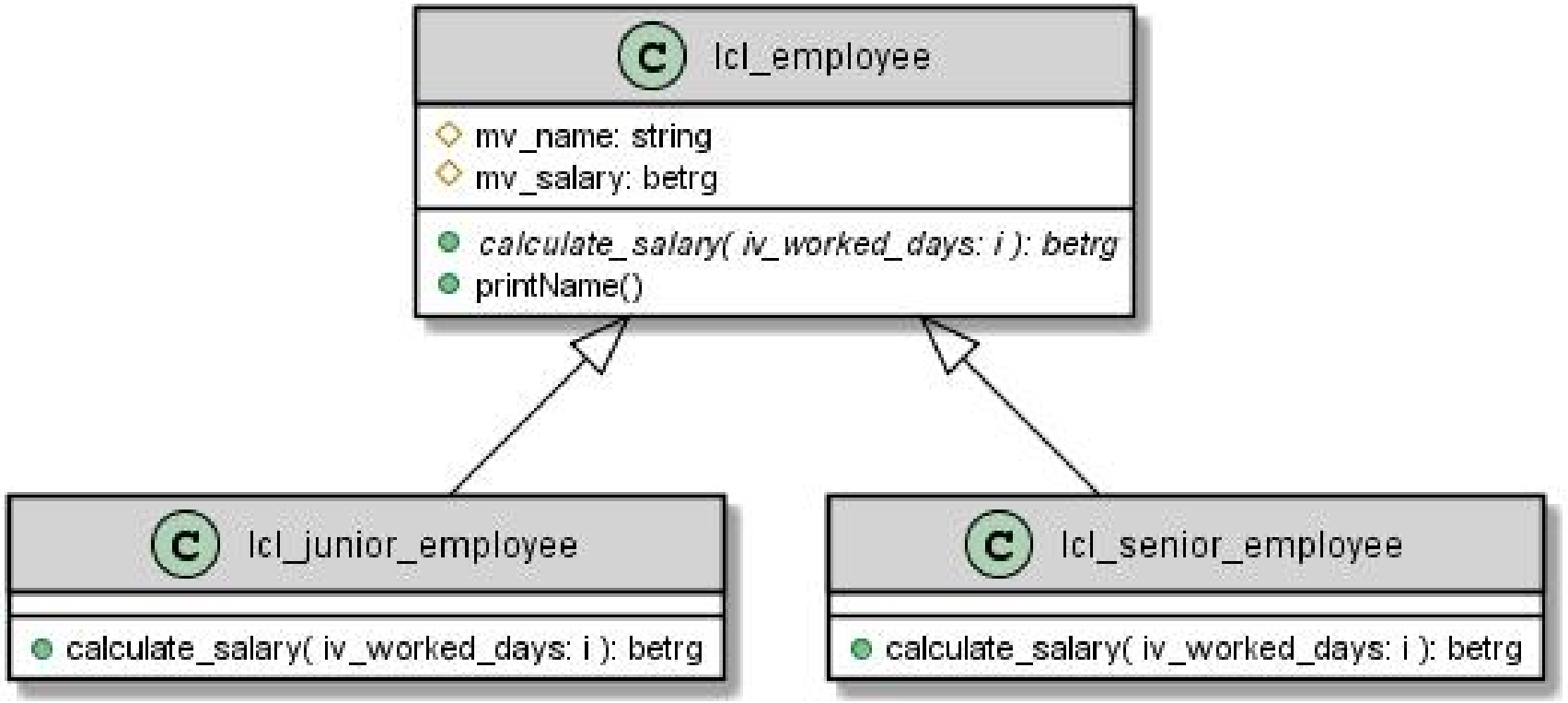
Interfaces

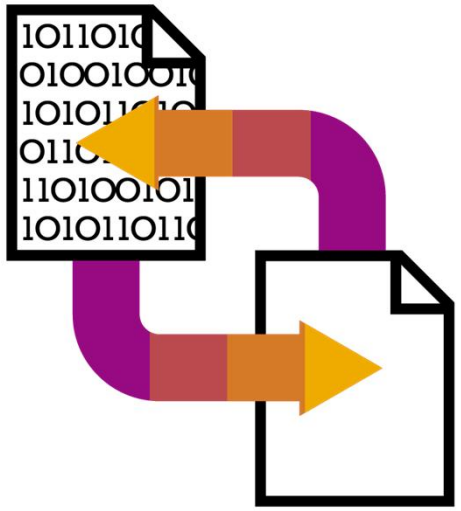
- A way of creating abstraction
- Defines a behavior without minding implementation
- Describes what an object can do
  - Also, describes what an object *must do* to be of a determined type;
- Creates a contract between the class and the outside world
  - An object that uses the class that implements the interface, only has to know the interface contract;
- A class that implements an interface **must implement all of its methods**
- Only **public** methods/attributes are defined on interfaces
- No implementation is allowed inside an interface definition



Abstract classes

- A class that has at least an abstract method
  - An abstract method is a method without implementation;
  - This method ***must*** be overwritten and implemented in the class' subclasses;
  - Private methods can't be abstract;
- An abstract class can't be instantiated
  - Only its non-abstract subclasses can be instantiated;

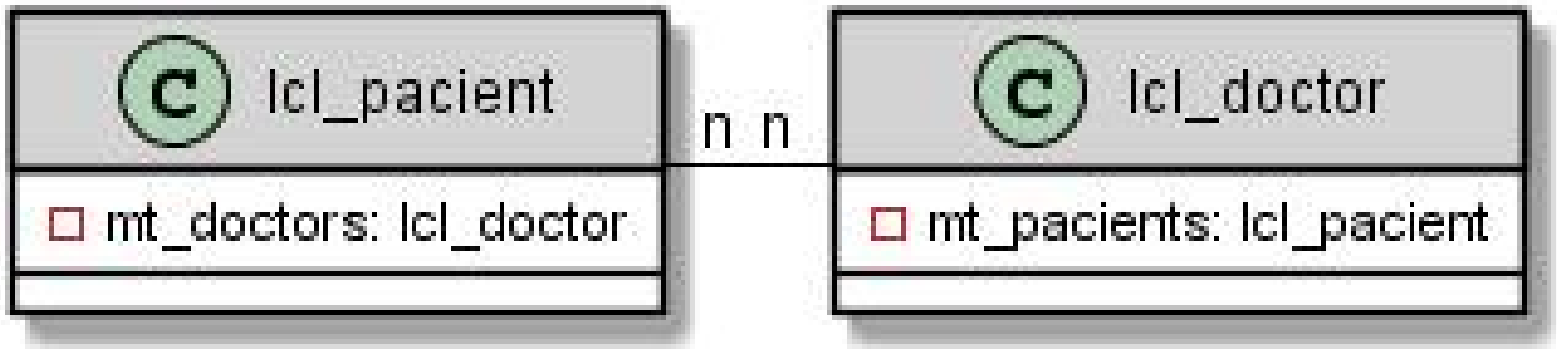




Relationships

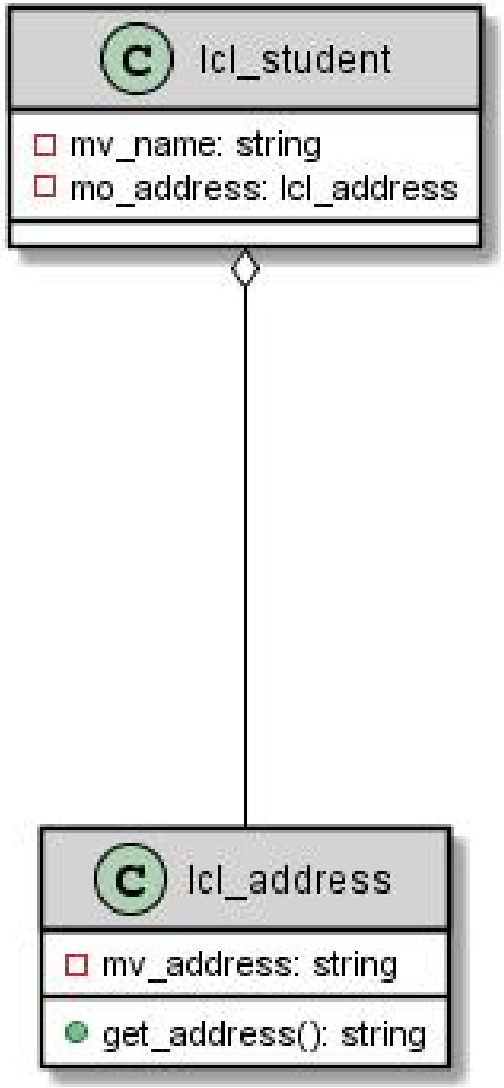
Association

- A ‘using a’ relationship type between two classes
  - One class uses another one’s object and vice-versa;
  - Both classes’ instances can exist independently in a meaningful way;
  - There isn’t an ‘owner’ class;



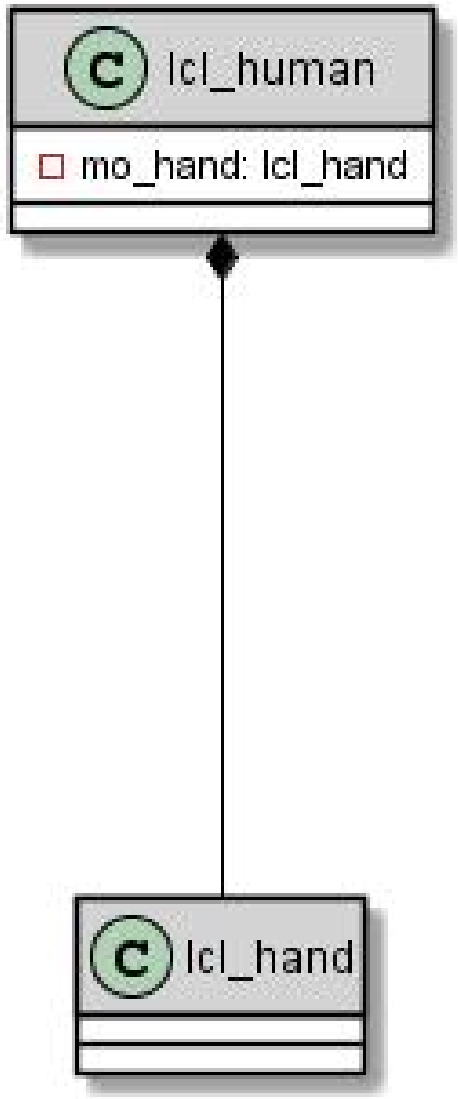
Aggregation

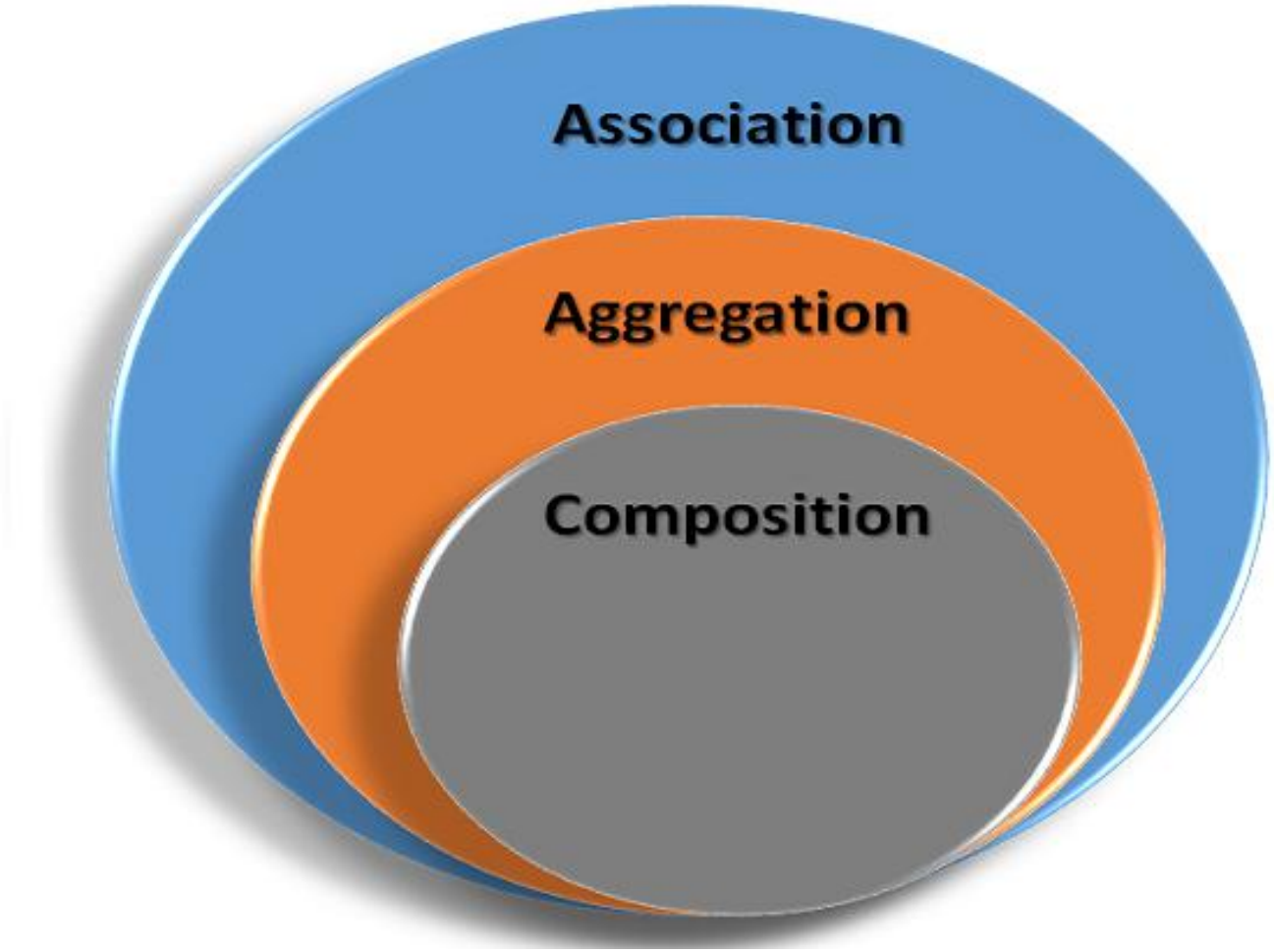
- A ‘has a’ relationship type between two classes
  - Special type of association;
  - Both classes’ instances can exist independently in a meaningful way;
  - One class works as an ‘owner’;
  - The owner uses the other class behavior;



Composition

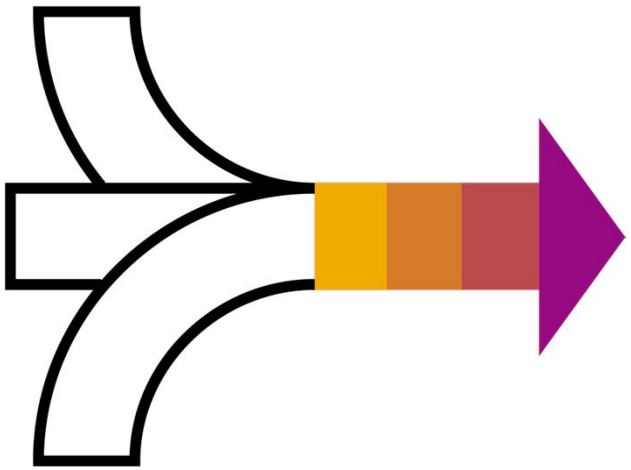
- Handles complexity
- A ‘part of’ relationship type between two classes
  - Another type of association;
  - Classes’ instances **can’t** exist independently in a meaningful way;
  - One class is the ‘owner’;
  - The owner needs the other class behavior;





HANDS ON

Part 1



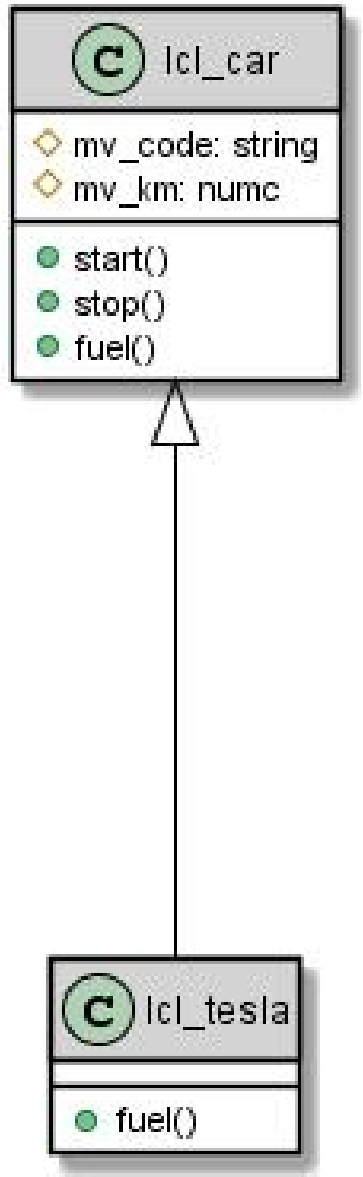
Inheritance

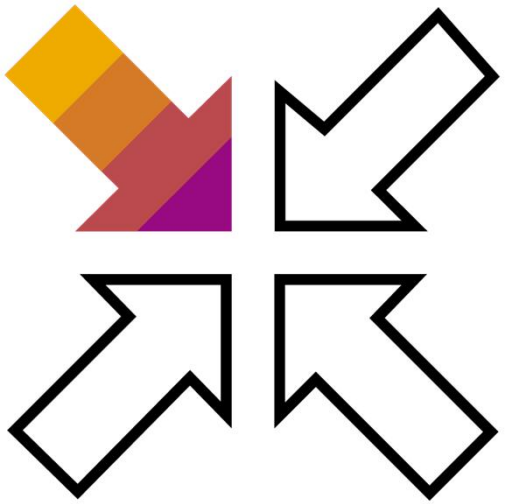
Inheritance

- A class (subclass) can reuse functionalities from another class (superclass) by inheriting them
  - A subclass can redefine a superclass method’s behavior, but not its signature;
  - Static methods can’t be redefined;
  - Private methods from superclass can’t be directly accessed nor redefined;
- A class can have many subclasses, but only one superclass
- A subclass can access all static non-private attributes of its superclass
- A subclass can implement other functionalities that aren’t included in the superclass

Inheritance

- When instantiating a subclass, we ***must*** guarantee that the superclass is being instantiated as well, by calling **super->constructor( )** inside the subclass constructor
- **Classes** defined as **final** can’t be inherited
- **Methods** defined as **final** can’t be redefined
- A subclass can access all static non-private attributes of its superclass
- A subclass can implement other functionalities that aren’t included in the superclass
- In a class both **final** and **abstract**, only static components can be used





Polymorphism

Polymorphism

- Write a code that works for your superclass
  - This code should also work for your subclasses;
  - Methods with the same name, but with different implementation for each subclass, should be called the same way;
- ABAP only supports dynamic polymorphism (via overriding)
- Other languages also have static polymorphism (via overloading)
  - Overloading is defining various methods with the same name, but with different signatures;
- Prefer composition over inheritance

Thread

Josh Justice
@CodingItWrong

Did you know that Beethoven's parents were rich but he had to turn down the family fortune to write music?

He preferred composition over inheritance.

5:21 PM · Jul 1, 2019 · Tweetbot for iOS

511 Retweets **1.3K** Likes

HANDS ON

Part 2

Thank you.

Contact information:

João Vitor de Camargo
Developer and Quality Engineer at HCM North America
joao.vitor.camargo@sap.com

Suzana Machado
Developer and Scrum Architect at HCM Spain
suzana.machado@sap.com