OBJECT ORIENTED PROGRAMMING

Class Relationship

by
Dr. Nor Saradatul Akmar Zulkifli
Faculty of Computer Systems & Software Engineering
saradatulakmar@ump.edu.my

Content Overview

- Unified Modelling Language (UML) Definition
- Type of UML Diagram
- Class Relationship
  - Generalization
  - Association
  - Dependency

Learning Objectives

Students should be able to:

- Understand the definition of UML and why UML is needed.
- Differentiate between types of class relationships.

U – UNIFIED

M – MODELING

L - LANGUAGE

WHY?

- To model software solution, application solutions, system behavior and business processes.

Structure Diagram

Things in the modeled system

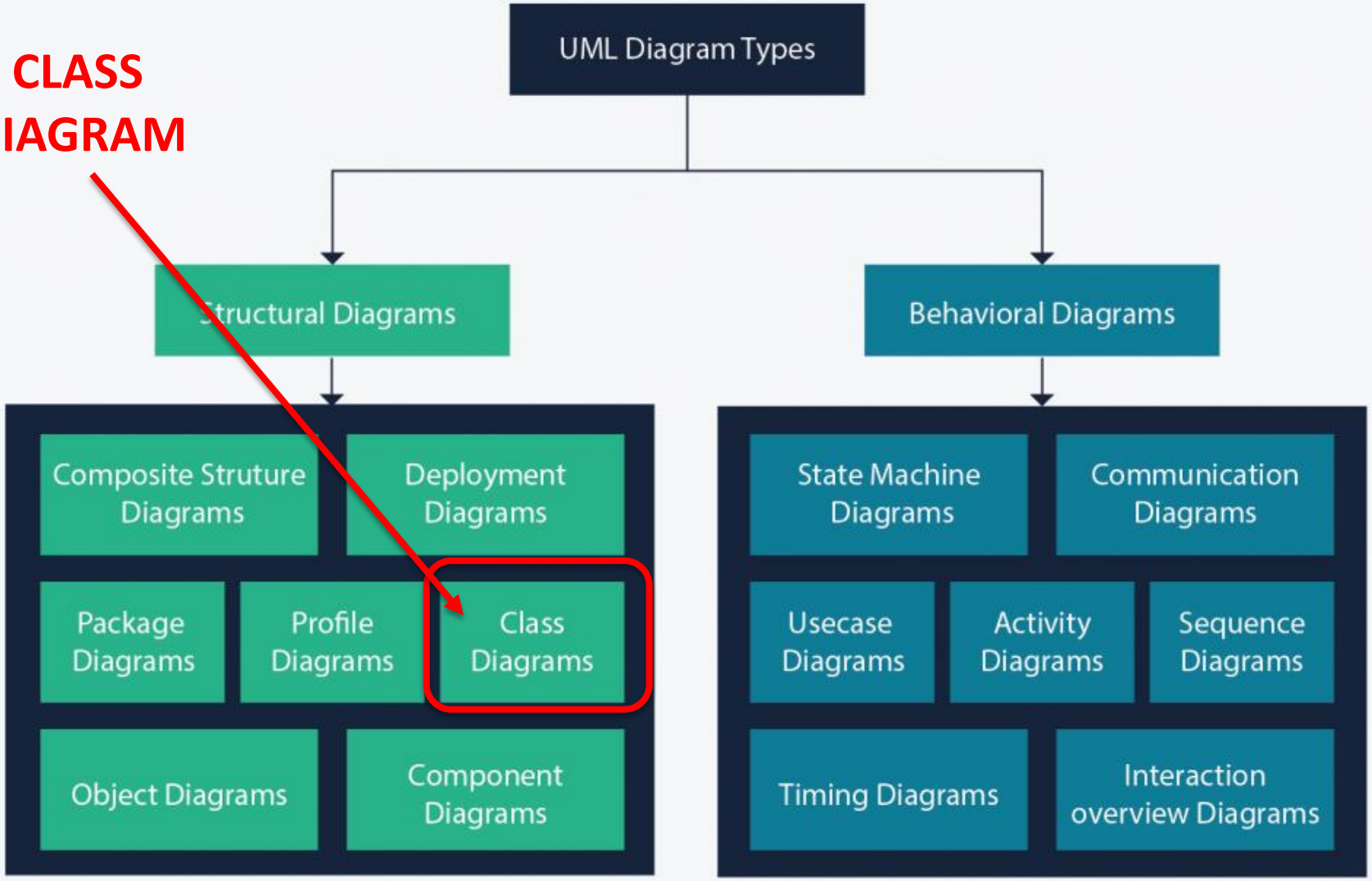
Technical Term : Show **different objects** in a system.

Behavior Diagram

What should **happen** in a system

Describe **HOW the objects interact** with each other to create a functioning system

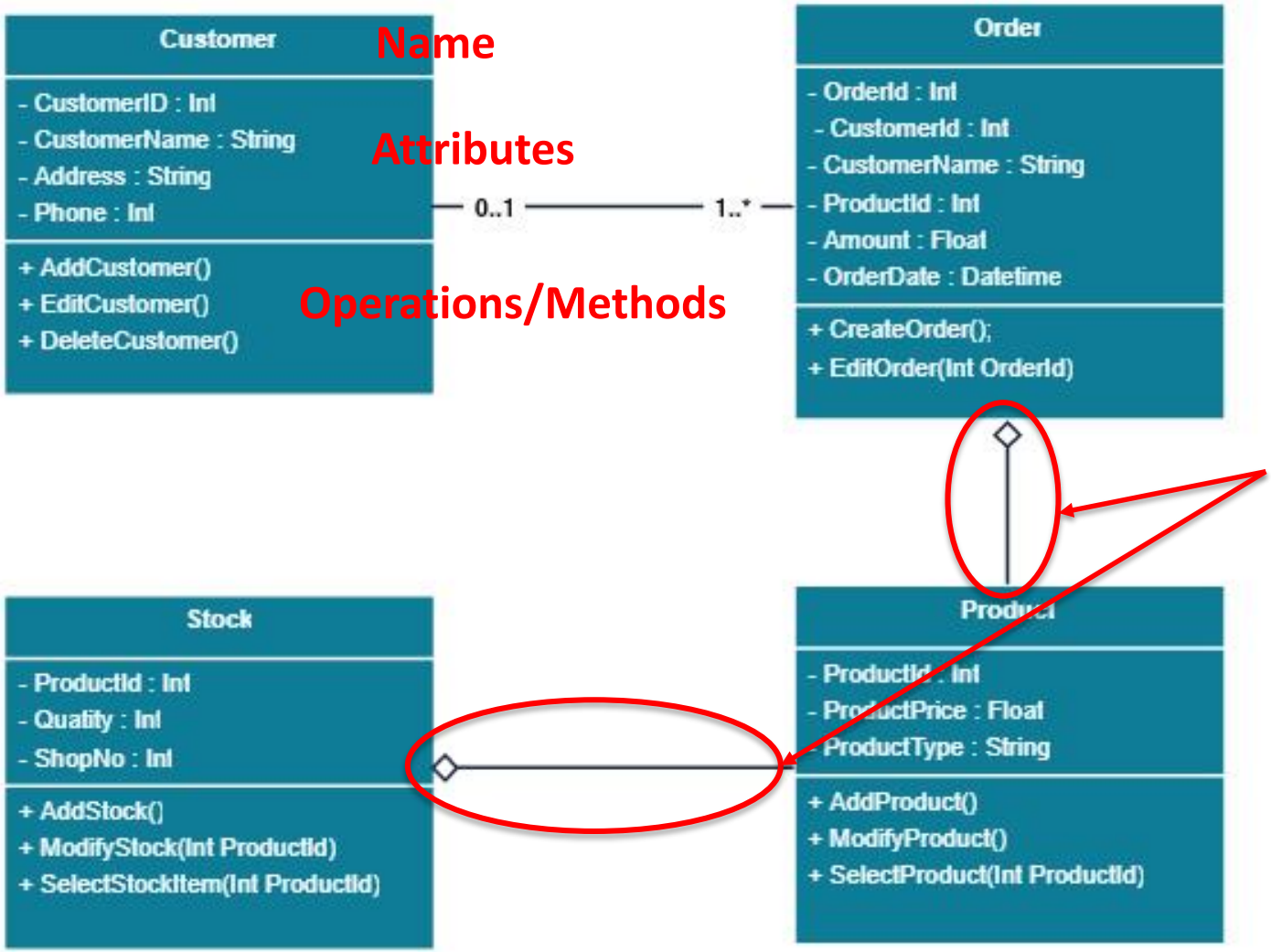
CLASS DIAGRAM



Source : http://creately.com/blog/diagrams/uml-diagram-types-examples/

EXAMPLE :

Class Diagram for Order Processing System



Source : http://creately.com/blog/diagrams/uml-diagram-types-examples/

CLASS DIAGRAM

- Describe the **types of objects** in the system and the various kind of ***static* relationships** that exist among them.
- Shows the **properties and operations of a class** & the constraints that apply to the way objects are connected.
- **Properties** – Structural features of a class compose of attributes and associations.

A class box has three parts:

CLASS ATTRIBUTES

- Attributes (fields, instance variables)
  - Visibility name: *type [count] = default_value*
  - Visibility : + public
    # protected
    - private
    ~ package (default)
    / derived
  - Derived attribute : not stored, but can computed from other attribute values
  - Underline : **static attributes**

Student

- name: String
- Id : int
- totalSTudents : int

getID ( ) : int
+ getName ( ) : String
~ getEmailAddress ( ) : String
+getTotalStudents ( ) : int

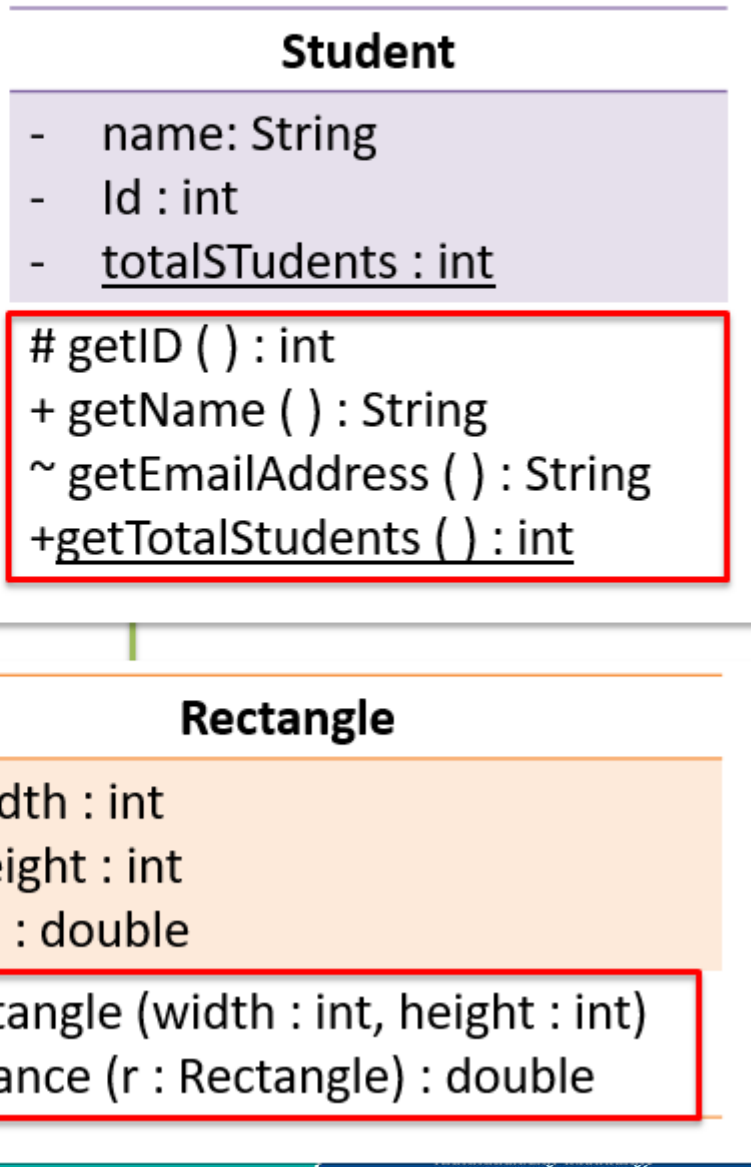
Rectangle

- width : int
- Height : int
/ area : double

+ Rectangle (width : int, height : int)
+ distance (r : Rectangle) : double

CLASS OPERATIONS / METHODS

- Operations / methods
  - *visibility name (parameters) : return_type*
  - visibility : + public
    # protected
    - private
    ~ package (default)
    / derived
  - Underline : **static methods**
  - Parameter types listed as (name:type)
  - Omit *return_type* for constructor and when return type is void
  - Method Example:
    + distance(p1:Point, p2:Point) : double



STATIC

Keyword : Static

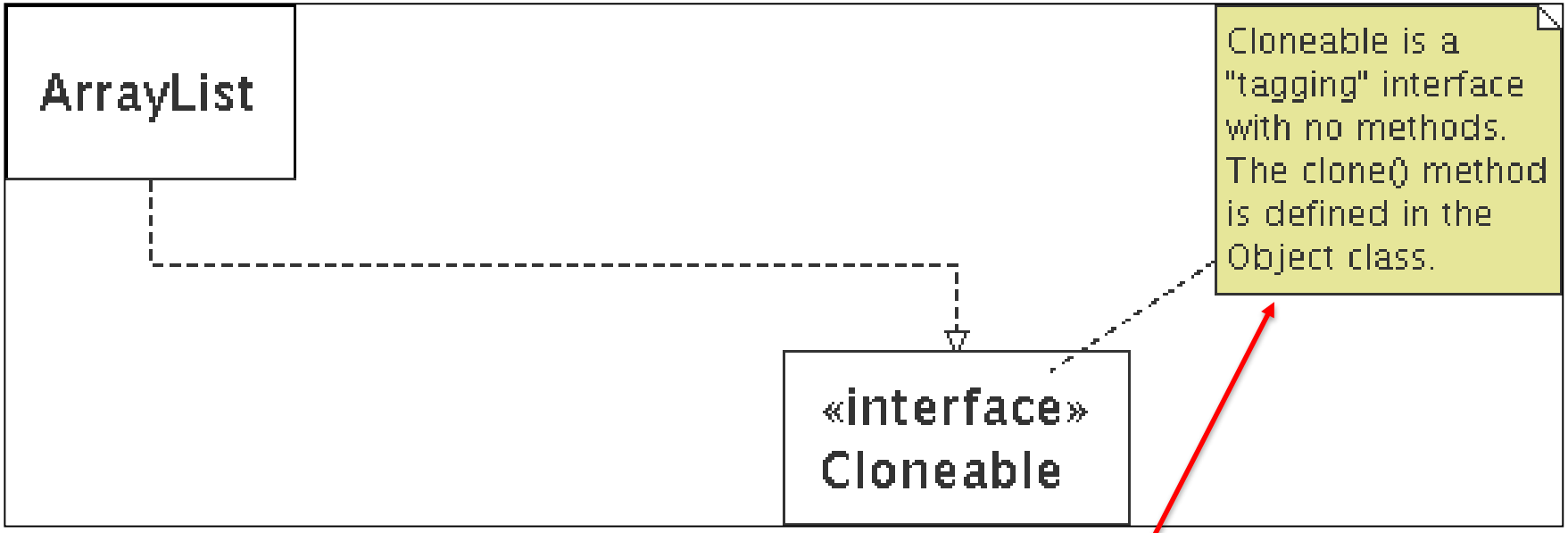
UML Class Diagram : **Underline**

One attribute for all objects of the class

Hint : No matter how many objects of the class are created, there is only one copy of the static member)

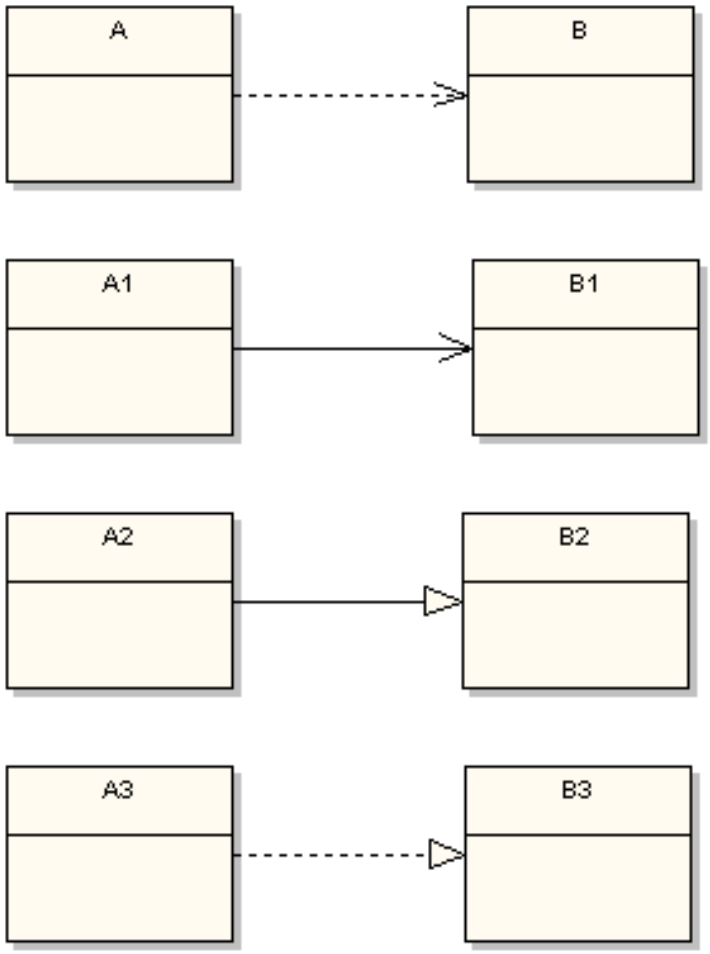
Static Class Methods – Have a class scope and they do not have access to the *this* pointer of the class.

COMMENTS



COMMENTS represented as a folded note, attached to the appropriate class/attribute/method by a dashed line

RELATIONSHIPS BETWEEN CLASSES

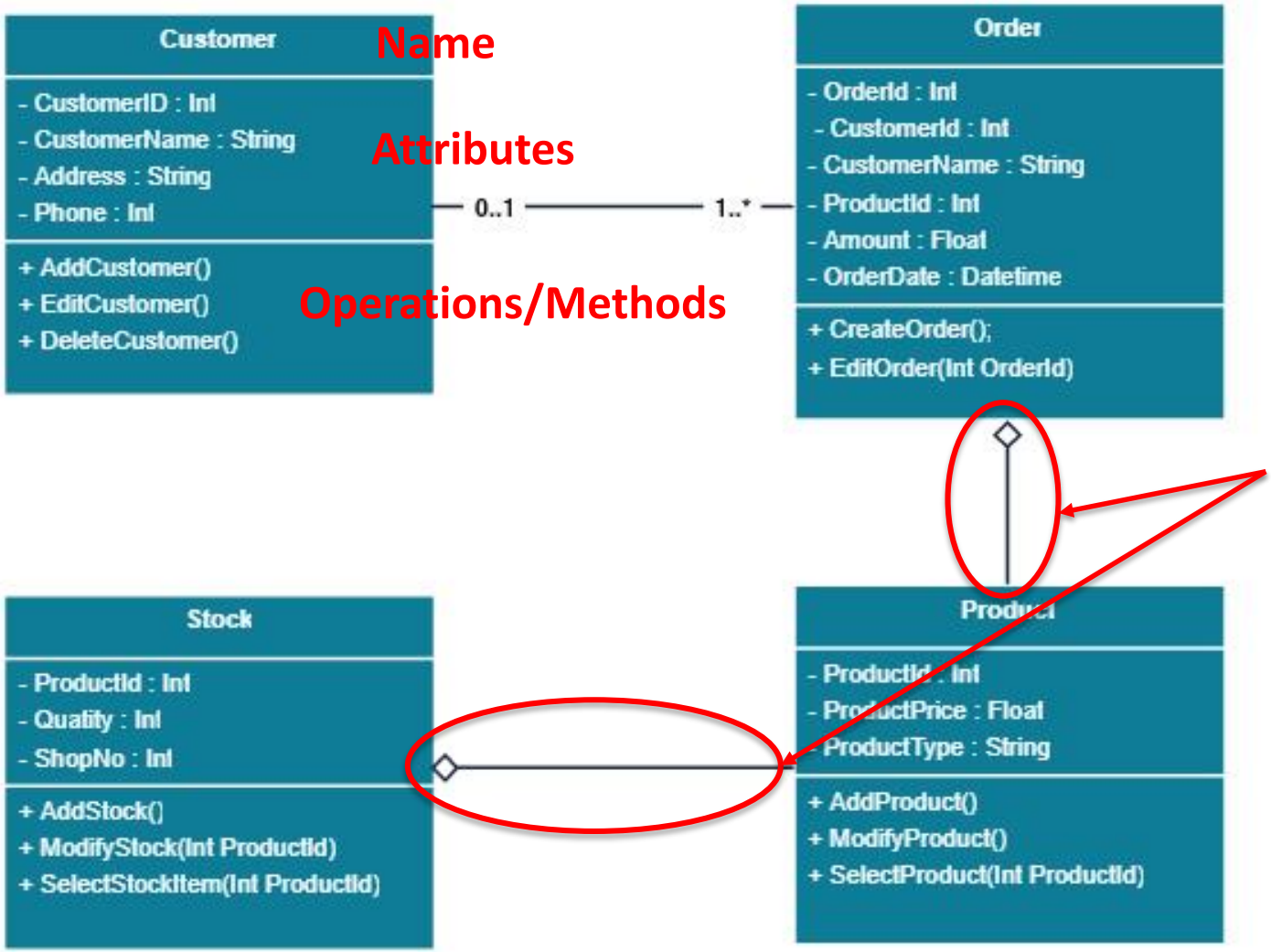


ARE THEY SHOW DIFFERENT RELATIONSHIP?

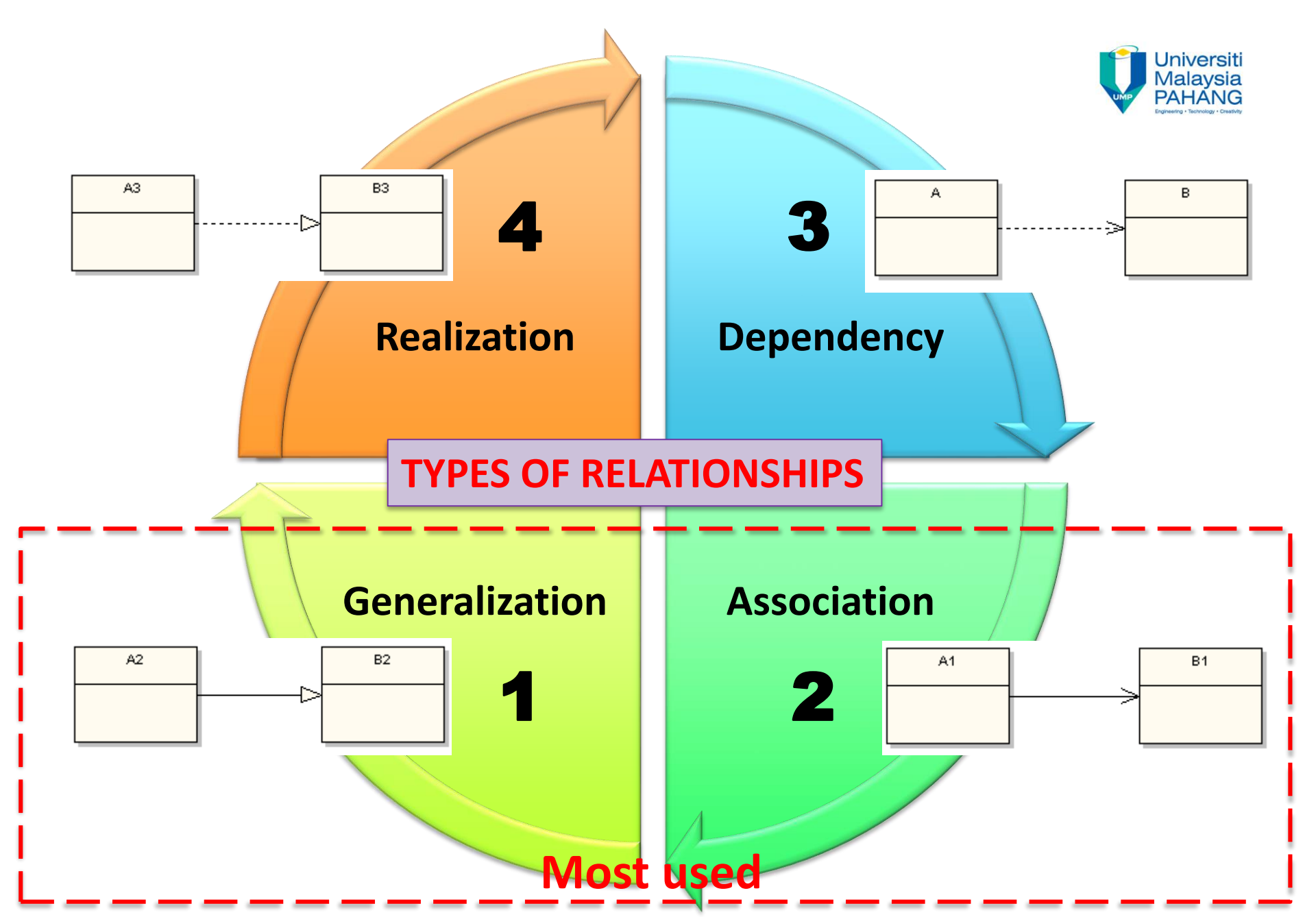
WHAT EACH OF THIS RELATIONSHIPS MEAN TO THE TARGET PROGRAMMING LANGUAGE (ex: JAVA)

EXAMPLE :

Class Diagram for Order Processing System



Source : http://creately.com/blog/diagrams/uml-diagram-types-examples/


A3
B3
4
Realization
3
Dependency
A
B
TYPES OF RELATIONSHIPS
Generalization
Association
A2
B2
1
2
A1
B1
Most used

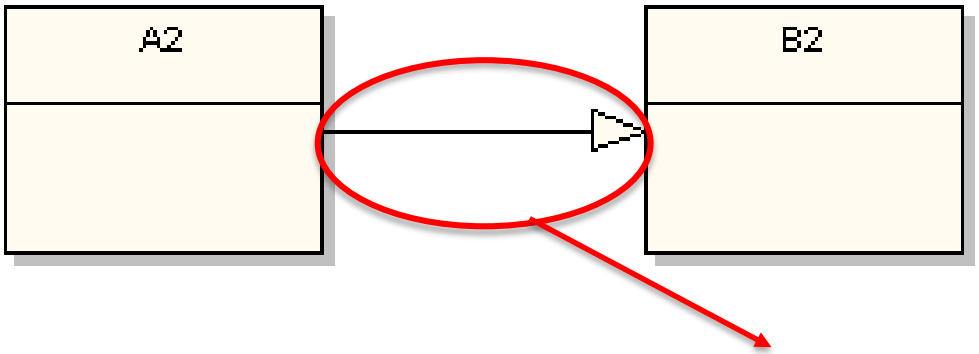
Generalization

- Often called as “IS A” (taxonomy) relationship
- Organizes classes into a hierarchy
- Implement inheritance between classes
- Implement interfaces

Associations

- (plain)Aassociation
- Aggregation
- Composition
- Dependency

1 GENERALIZATION



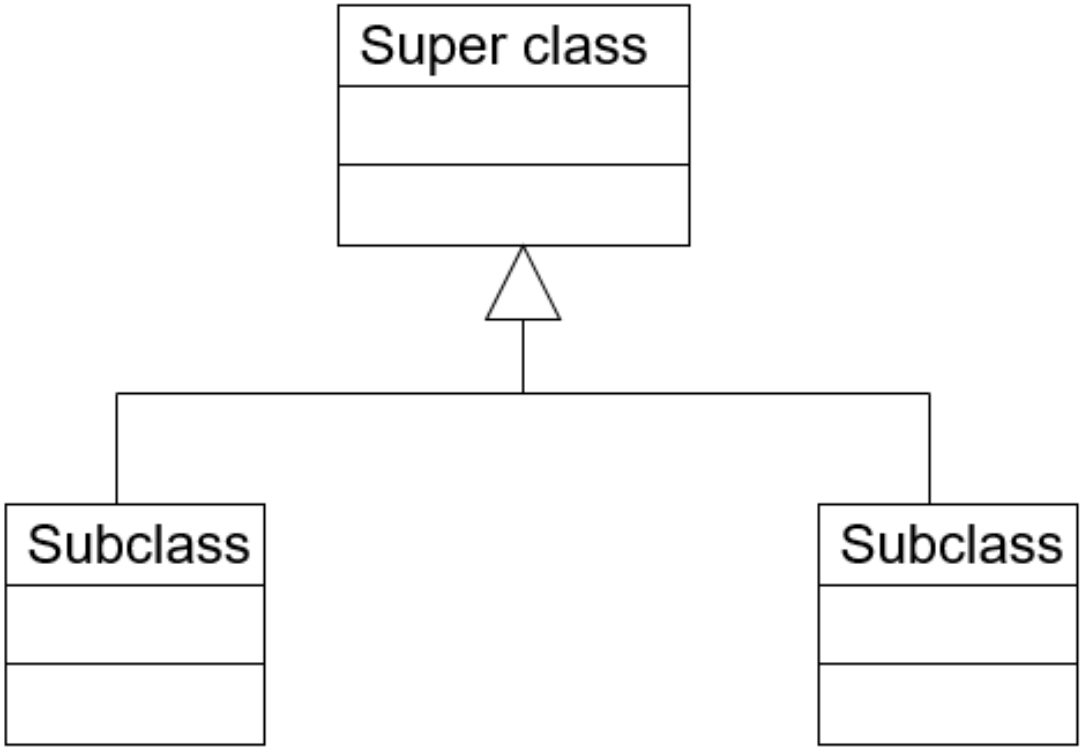
EXAMPLE:

- B2 is the super class and A2 is the subclass in the relationship

```
import B2;

public class A2 extends B2 {
    // . . .
    }
```

1 GENERALIZATION



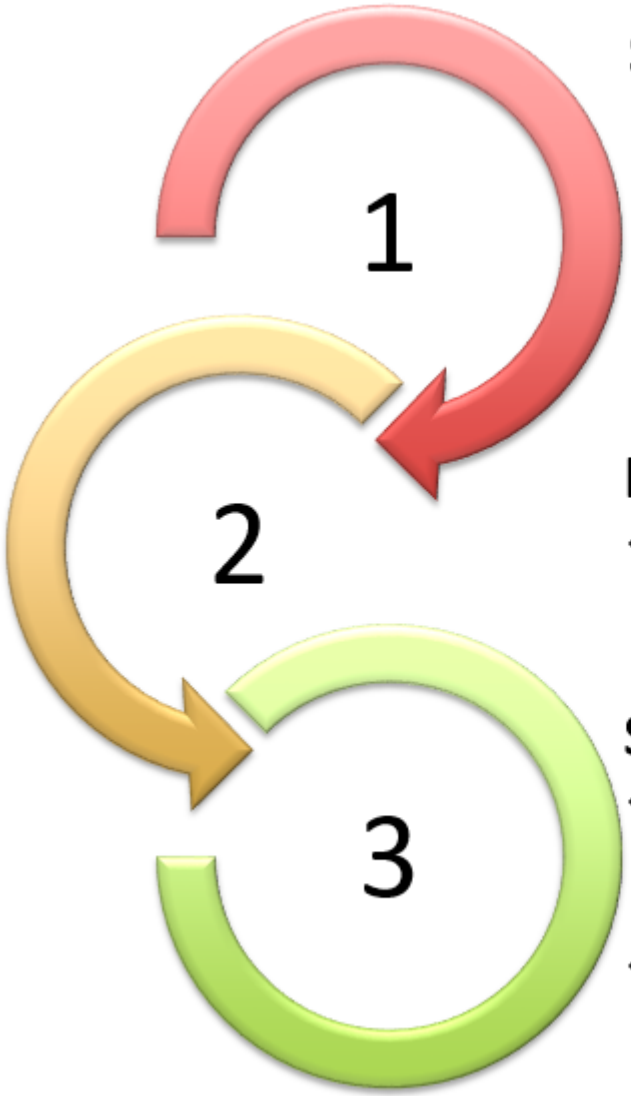
- Shows **inheritance** between a super class and a subclass
- Is represented by **a line with a clear triangle arrowhead** pointing at the super class

1 GENERALIZATION

- Generalization = The relationship between a class (superclass) and one or more variations of the class (subclasses)
- It's organizes classes by their similarities, structuring the descriptions of objects
- A superclass holds common attributes, attributes and associations
- The subclasses
  - ✓ inherit the features of their superclass
  - ✓ Adds specific attributes and operations
  - ✓ May override a superclass attributes and operations by redefining with the same name
  - ✓ Not allowed to override the signature (name, amount and type of parameters) or methods.

Inheritance will be discuss further in Week 6 : Inheritance

1 GENERALIZATION : WHY?



Structuring the descriptions of objects:

- Forming a taxonomy (classification), organizing objects according to their similarities.
- More effective that modeling each class individually in isolation of other similar classes.

Enabling code reuse:

- By reusing the code, its more productive than repeatedly writing code from scratch.

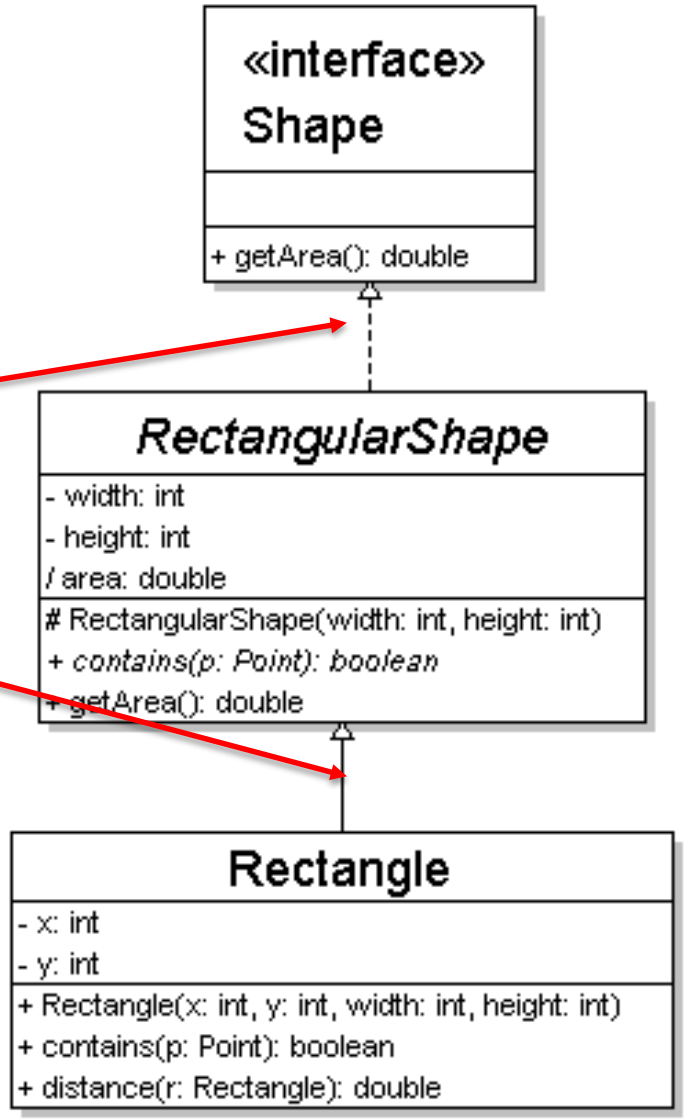
Support of Polymorphism:

- Subclass automatically inheriting superclass behavior
- Overriding and Polymorphism increases the flexibility of code

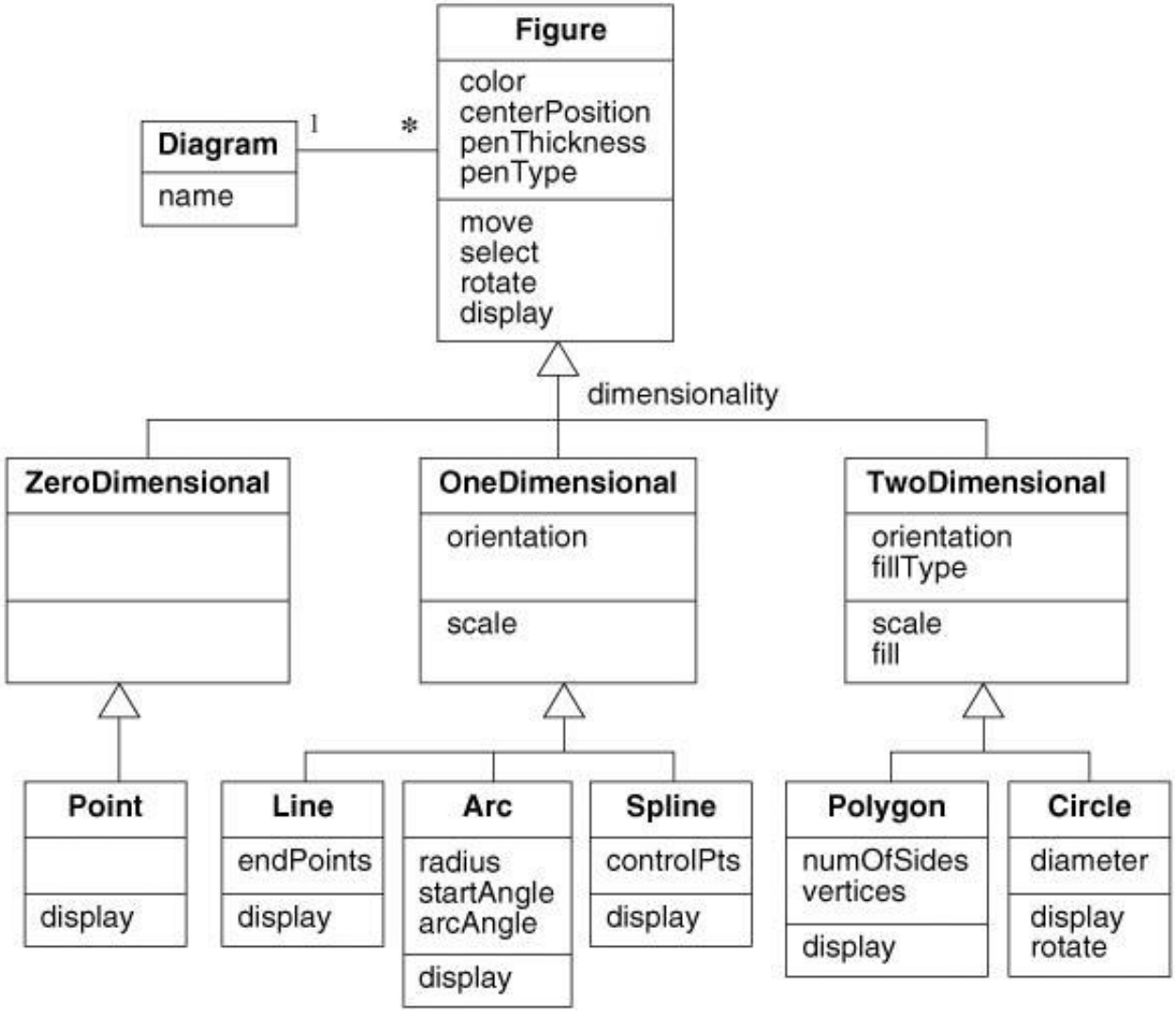
1 GENERALIZATION : HOW?

Generalization Relationships:

- Hierarchies to be drawn top-down with arrows pointing up to the parent (superclass)
- **Line/arrow styles** are differ, based on whether parent is a(n):
  - **Interface = dashed line, white arrow**
  - **Class and abstract class** = solid line, white arrow



1 GENERALIZATION : EXAMPLE



1 GENERALIZATION & INHERITANCE

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

Inheritance can be model "early" but not implement it.

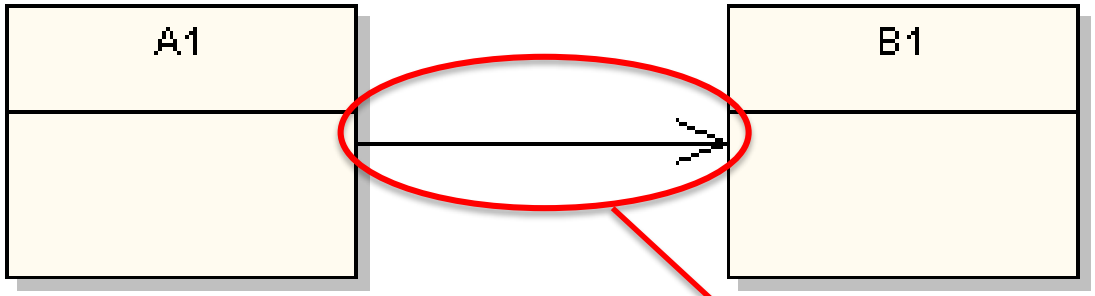
- ❑ Generalization represents a relationship at the **conceptual** level
- ❑ Inheritance is an **implementation** technique

Inheritance may not be appropriate when it's time to implement

- ❑ Objects should never change from one subclass to another
- ❑ *Composition* can be used instead

No Multiplicities for generalization are marked.

2 ASSOCIATIONS



There is a one-way relationship

EXAMPLE:

- Class A1 associated with class B1
- Class A1 uses and contains one instance of class B1, BUT B1 does not know about or contain any instances of class A1.

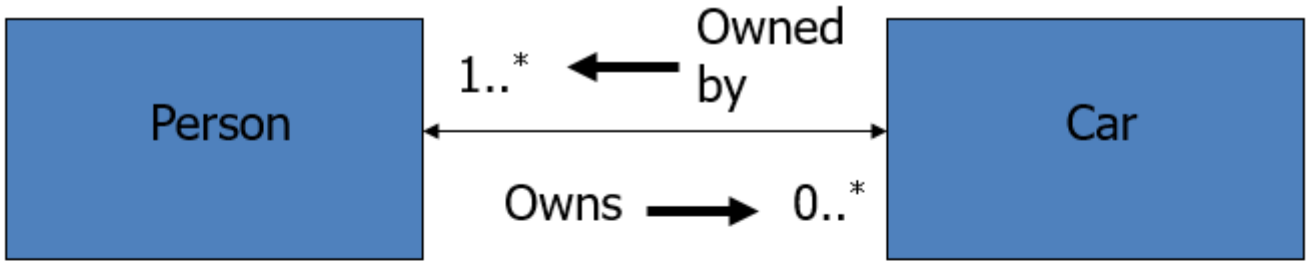
```
import B1;
public class A1 {
    private B1 b1;
    public B1 getB1 ()
    {
        return b1;
    }
}
```

2 ASSOCIATIONS : USAGE

Classes A and B are associated if:

- An object of class A sends a message to an object of B
- An object of class A creates an instances of class B
- An object of class A has an attribute of type B or collections of objects of type B
- An object of class A receives a message with an argument that is an instance of B

2 ASSOCIATIONS : EXAMPLE

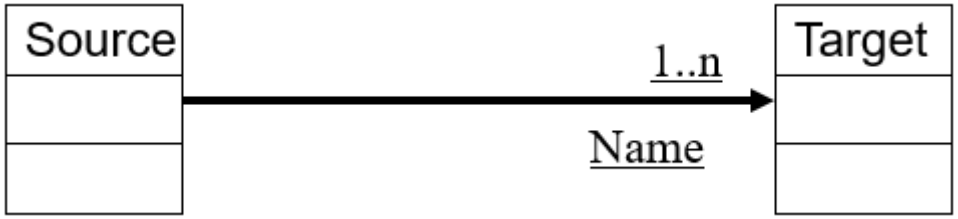


- A person own 0-many cars. A car is owned by 1 to many people
- If power of relationships are not specified, then it assumed to be 1-1.
- Relationships can begin and end with any number . . . i.e. – can have 5. . 11 or 19 . . 25
- Relationships can be multiple list – (1,4,6,8 . . 12)

2 ASSOCIATIONS

Associations relationships:

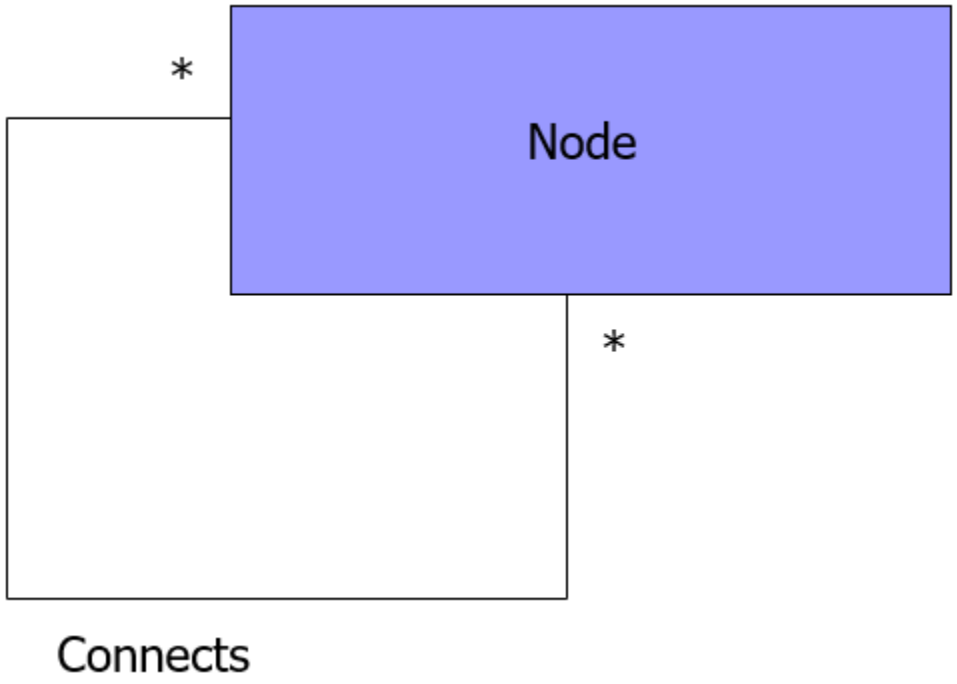
- A connection between classes
- It define the state of instances of dependent class
- Represented by a solid line between two classes directed from the source class to the target class.
- Multiplicities shown at both ends (USUALLY at the target end)
- Arrow either one or bidirectional



2 ASSOCIATIONS : RECURSIVE

Class that use itself

Example : Network Nodes (Class Diagram)

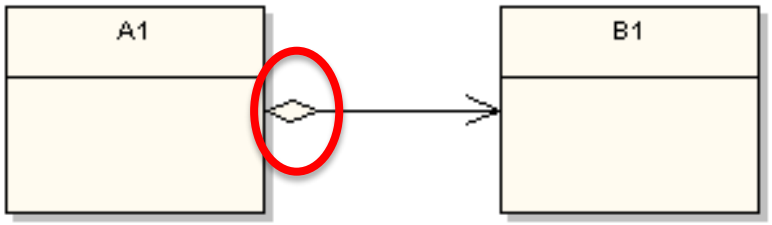


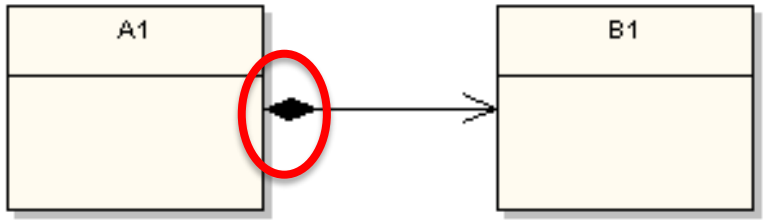
2 ASSOCIATIONS : SPECIAL TYPES

AGGREGATION

COMPOSITION

UML Symbol of **a diamond** at the source end of a line

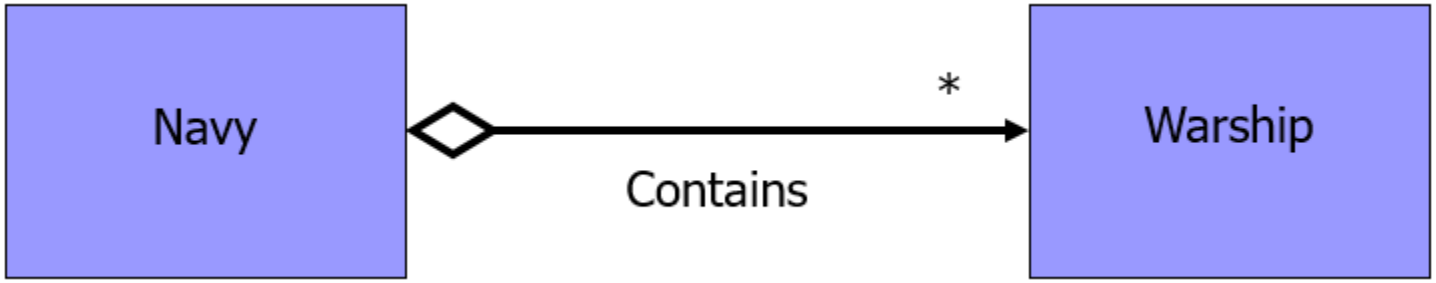




An association where an instance of A1 **contains a reference to an instance of B1** as part of the A1's state, BUT the use of the specific instance of B1 is or may be shared among aggregators.

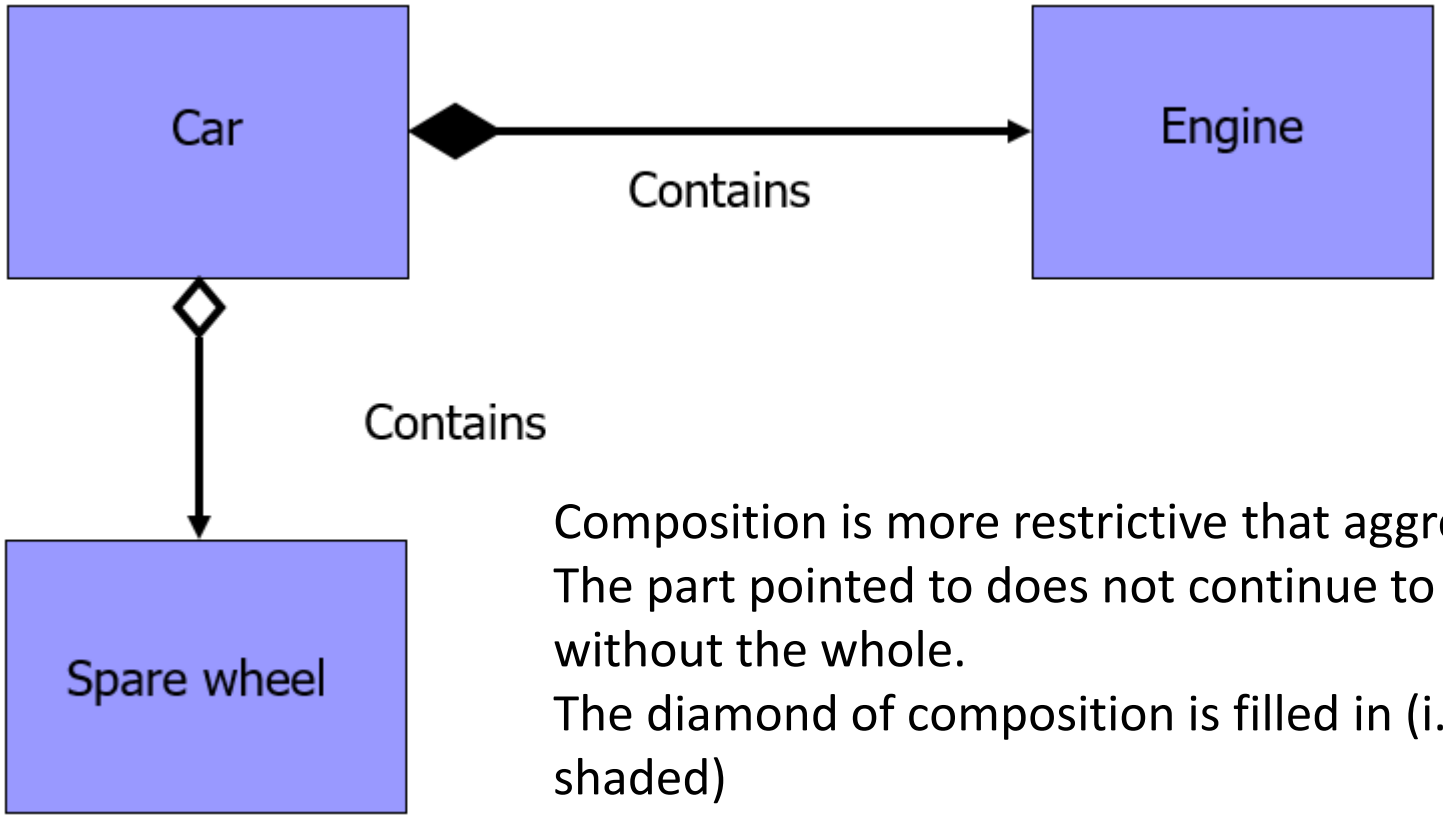
A relationship where **the scope** of the containing object (an A1) and the contained object (a B1) is **related**.

2 AGGREGATION : EXAMPLE



The navy contains warships.
Warships can be added or removed, but you still have a Navy.
The hollow diamond shows aggregation.

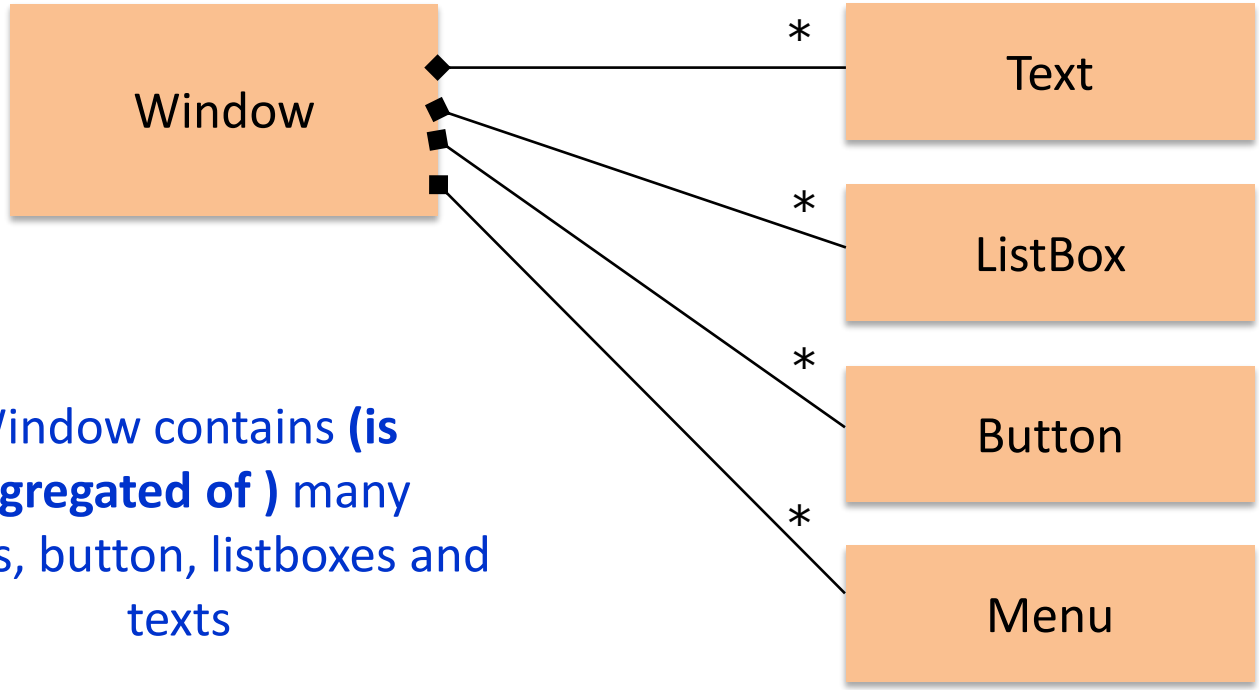
2 AGGREGATION vs. COMPOSITION



Composition is more restrictive that aggregation.
The part pointed to does not continue to exist without the whole.
The diamond of composition is filled in (i.e. shaded)

2 COMPOSITION

- An aggregation that owns its parts
- Its strong ownership

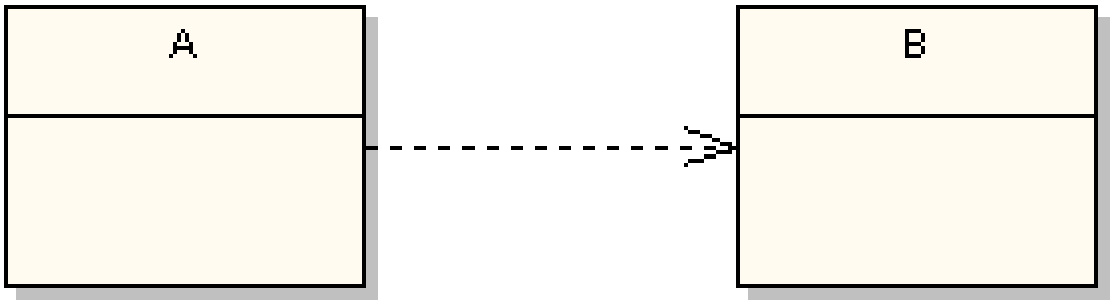


Window contains **(is aggregated of)** many menus, button, listboxes and texts

ASSOCIATIONS VS. GENERALIZATION

See The Differences??

3 DEPENDENCY



- A dependency exists between two elements if **changes of one element (the server) may cause changes to the other element** (the client)
- Once a dependency is created, if a class changes its interface, any message sent to that class may no longer be valid.
- **Examples**:
  - One class accesses a global object of another class
  - One class calls a method in another class
    - One class sends a message to another class
    - One class mentions another as a parameter of an operation

3 DEPENDENCY : EXAMPLE

- The relationship is shown as a dashed line with an arrow.
- The type of dependency is called a stereotype – In this case, it is a friend dependency
- **GENERAL RULE : MINIMIZE DEPENDENCIES**

CLASS DIAGRAMS : WHEN?

Class diagram are the **backbone of UML** and are the most used diagrams

Normal situation:

Use only a subset of the notations (class box with name, attributes, operations, association, aggregation and generalization)

Class diagrams only model software structure. Thus, get too focused on class diagram and ignore behavior.

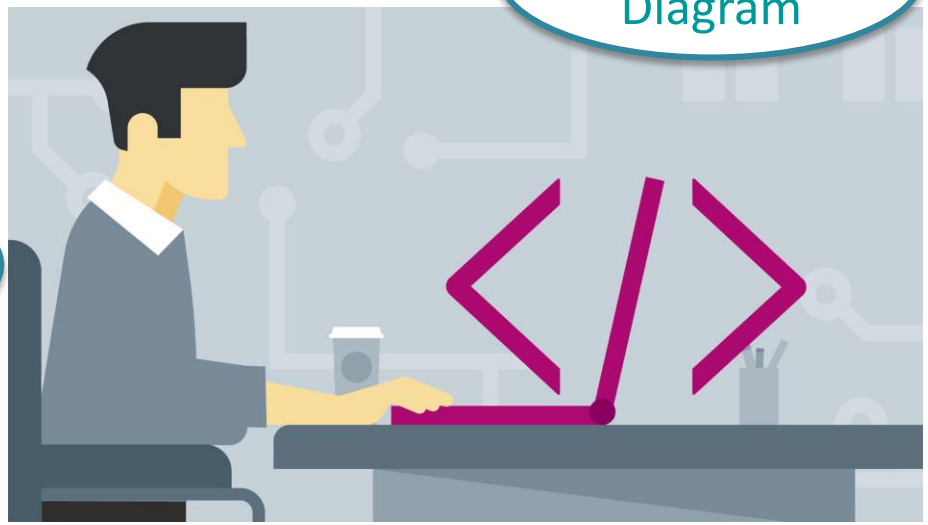
TASK:

EXPLORE THE OTHER TYPE OF DIAGRAMS

Object Diagram

Use Case Diagram

Activity Diagram



Author Information

Dr. Nor Saradatul Akmar Binti Zulkifli

Senior Lecturer
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang