

CLASS DIAGRAM

- Class diagram is a static diagram.
- It represents the static view of an application.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Purpose of Class Diagrams

- Shows static structure of classifiers in a system
- Diagram provides a basic notation for other structure diagrams prescribed by UML
- Helpful for developers and other team members too
- Business Analysts can use class diagrams to model systems from a business perspective

CLASS DIAGRAM

- A UML class diagram is made up of:
- A set of classes and
- A set of relationships between classes

CLASS DIAGRAM

- Each class is represented by a rectangle having a subdivision of three compartments:
 1. name (class name)
 2. attributes (variables)
 3. Operations (methods)

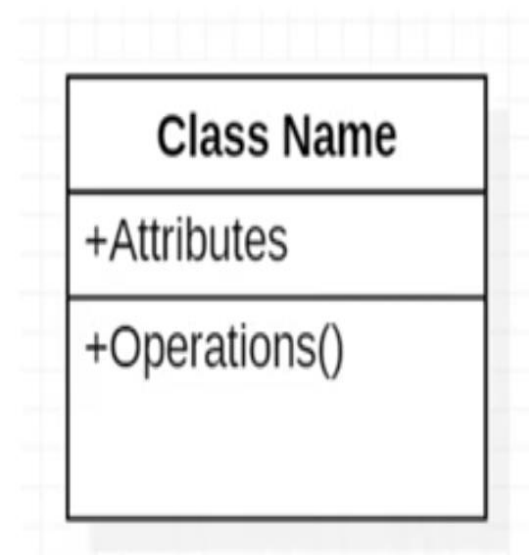
- There are three types of modifiers which are used to decide the visibility of attributes and operations.
 - + is used for public visibility
 - # is used for protected visibility
 - – is used for private visibility

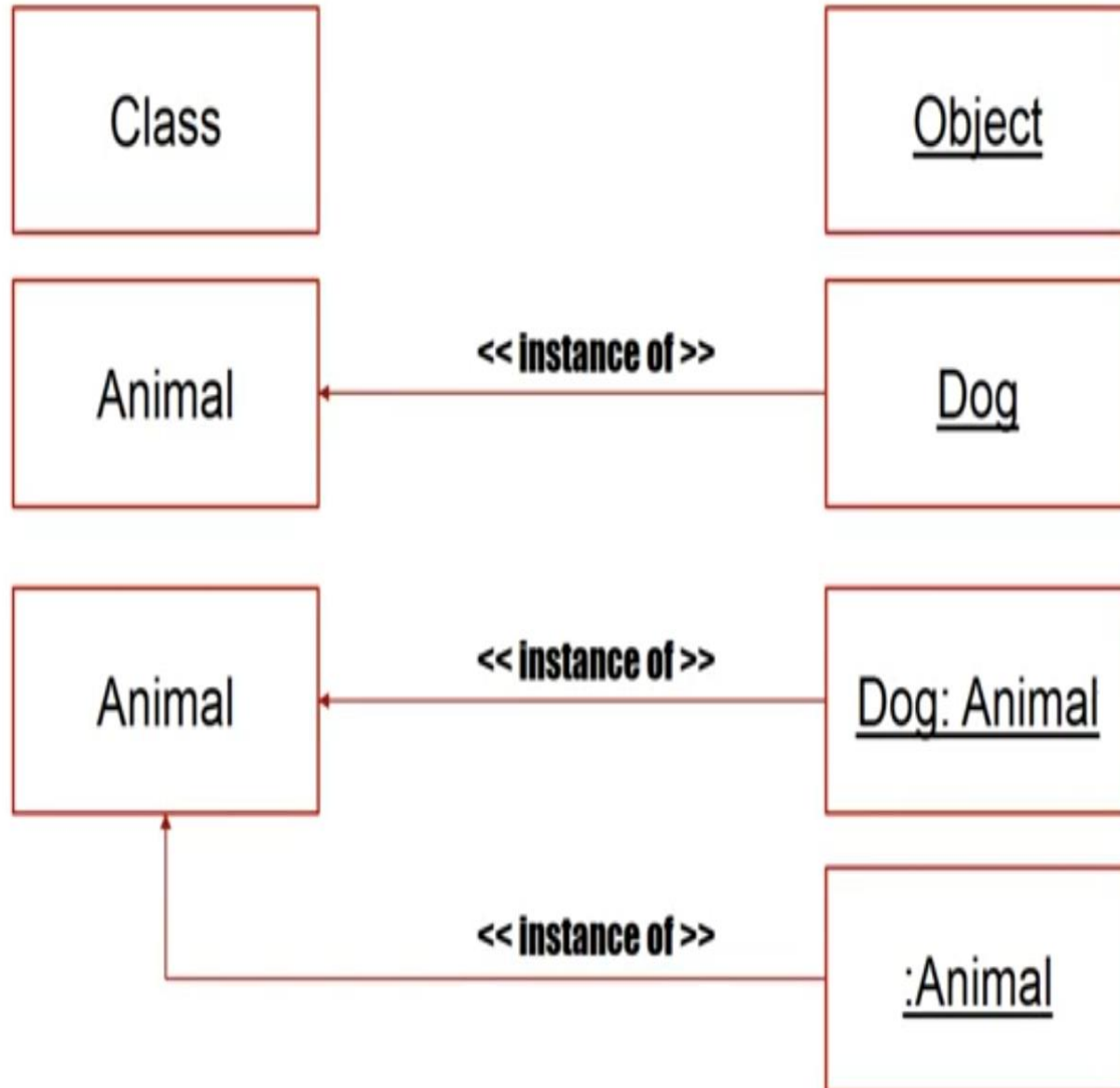
CLASS

- CLASS IS A TEMPLATE OR FORMAT.
- REPRESENT ENTITIES WITH COMMON CHARACTERISTICS OR FEATURES.

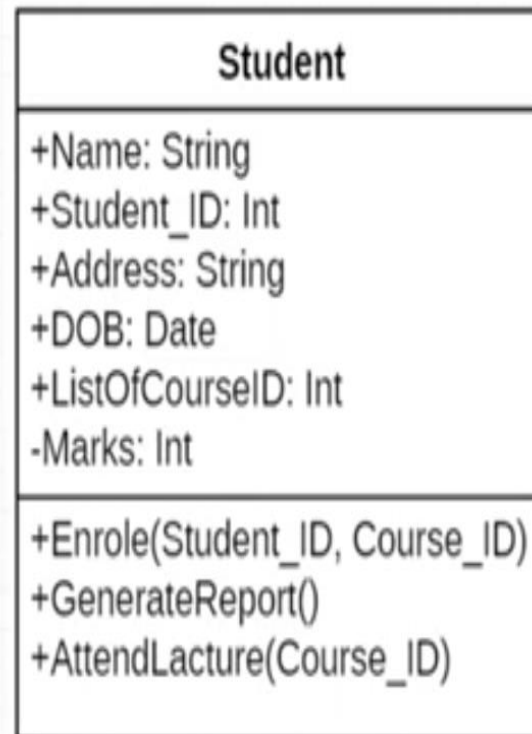
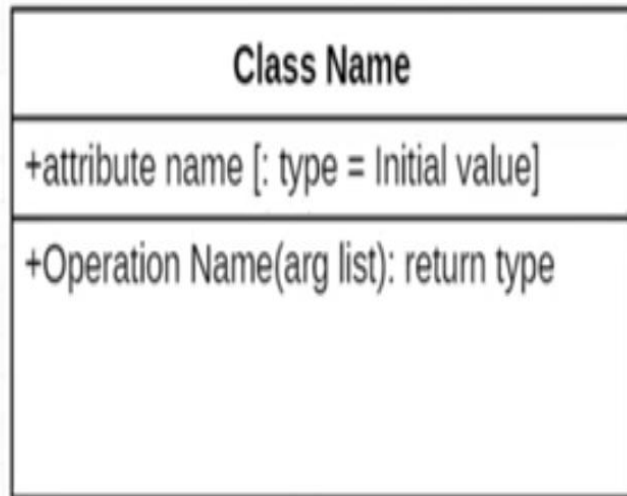
THESE FEATURES INCLUDE:

- ☐ ATTRIBUTES - DATA
- ☐ OPERATIONS - METHODS
- ☐ ASSOCIATIONS





CLASS REPRESENTATION



← Class Name

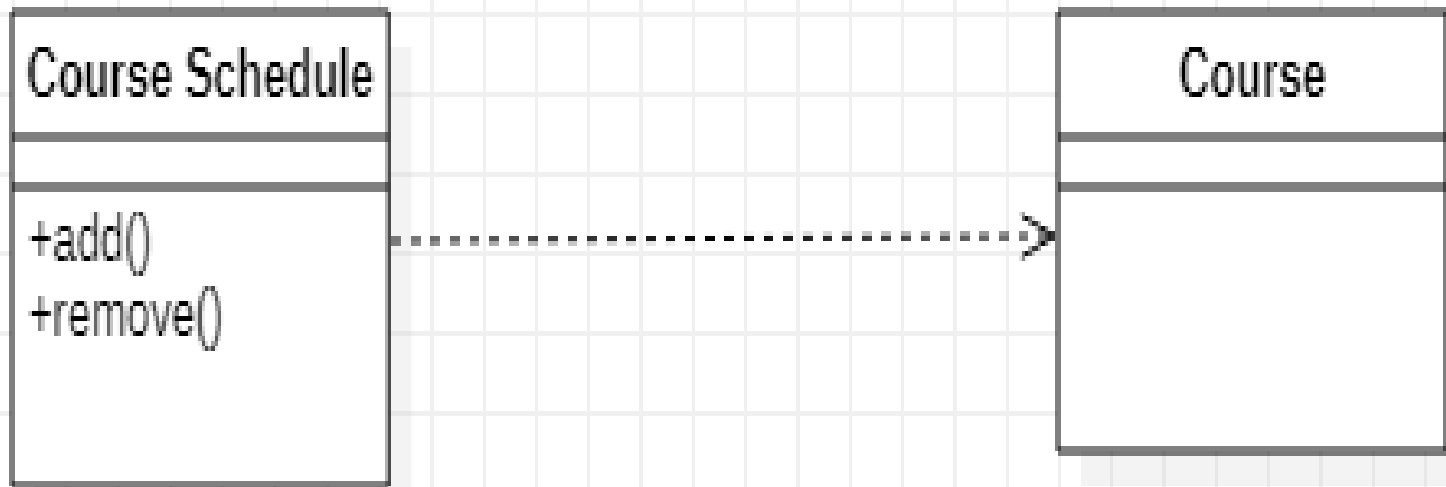
← Attributes

← Operations

Class Relationships

1. Dependency - Class1 depends on Class2

A dashed line with an open arrow



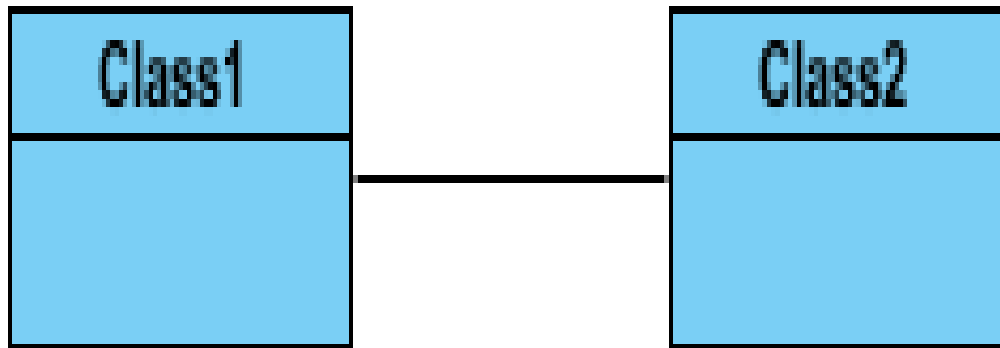
2.Inheritance (or Generalization):



3. Association

➤ Simple Association:

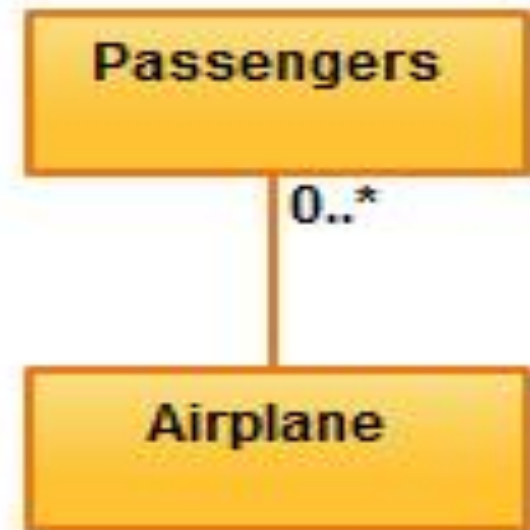
- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



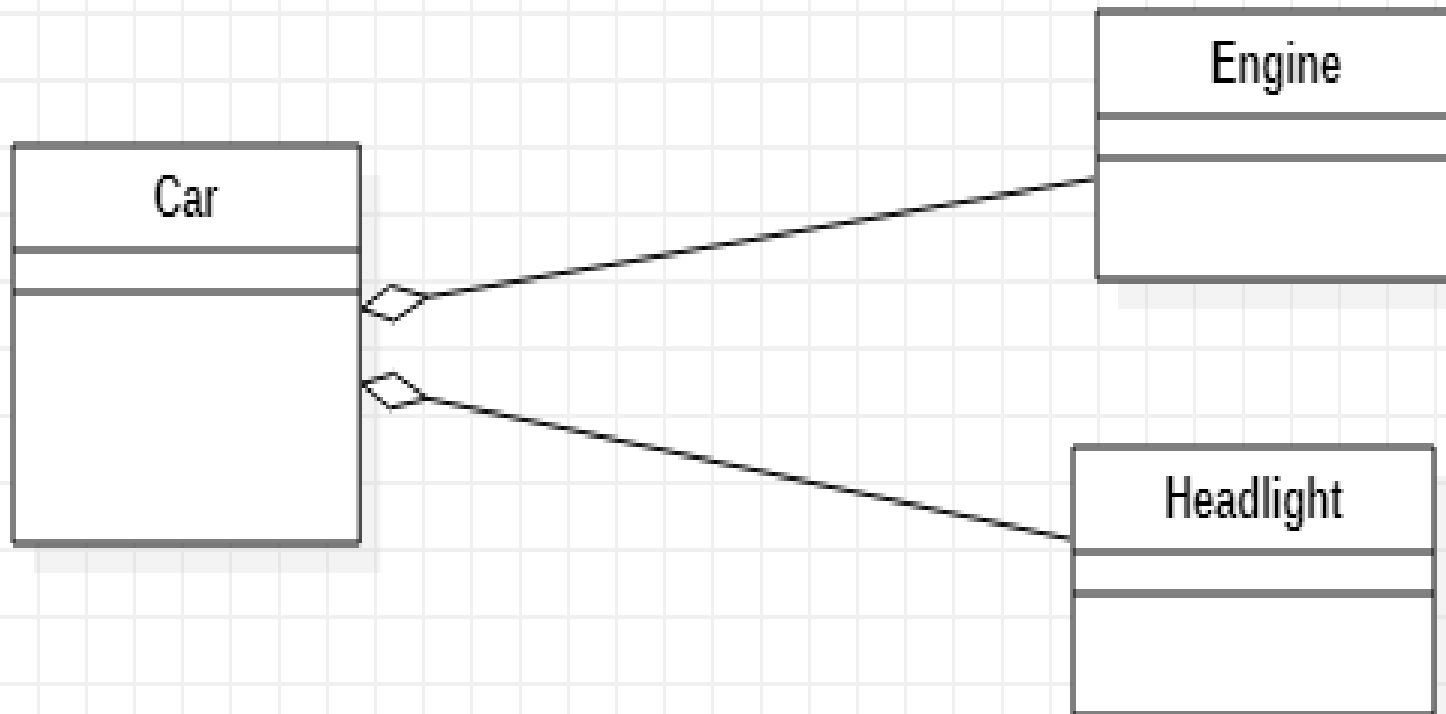
4. Multiplicity - is the active logical association when the cardinality of a class in relation to another is being depicted.

multiplicity can be expressed as:

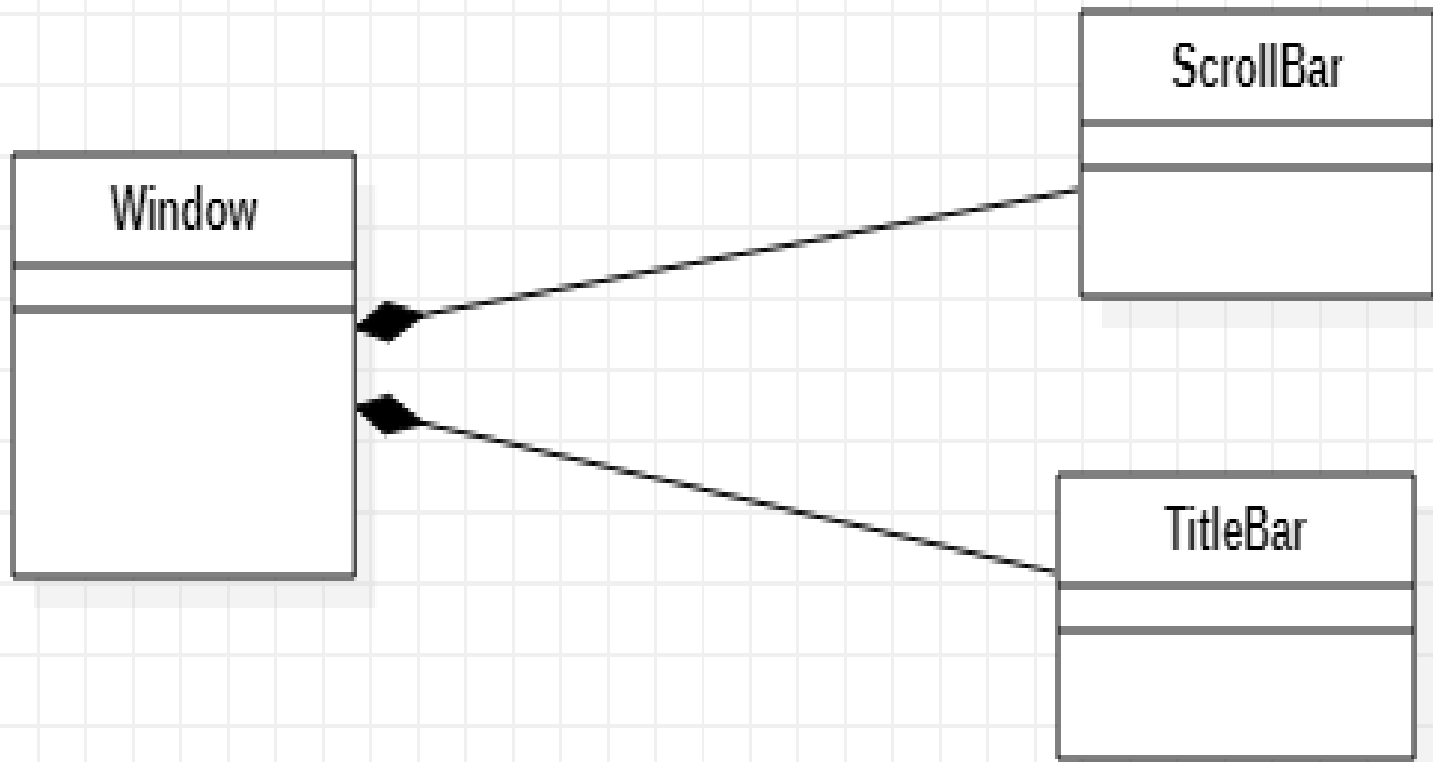
- Exactly one - 1
- Zero or one - 0.....1
- Zero or more - 0.....*
- One or more - 1.....*
- Specified range 2.....4

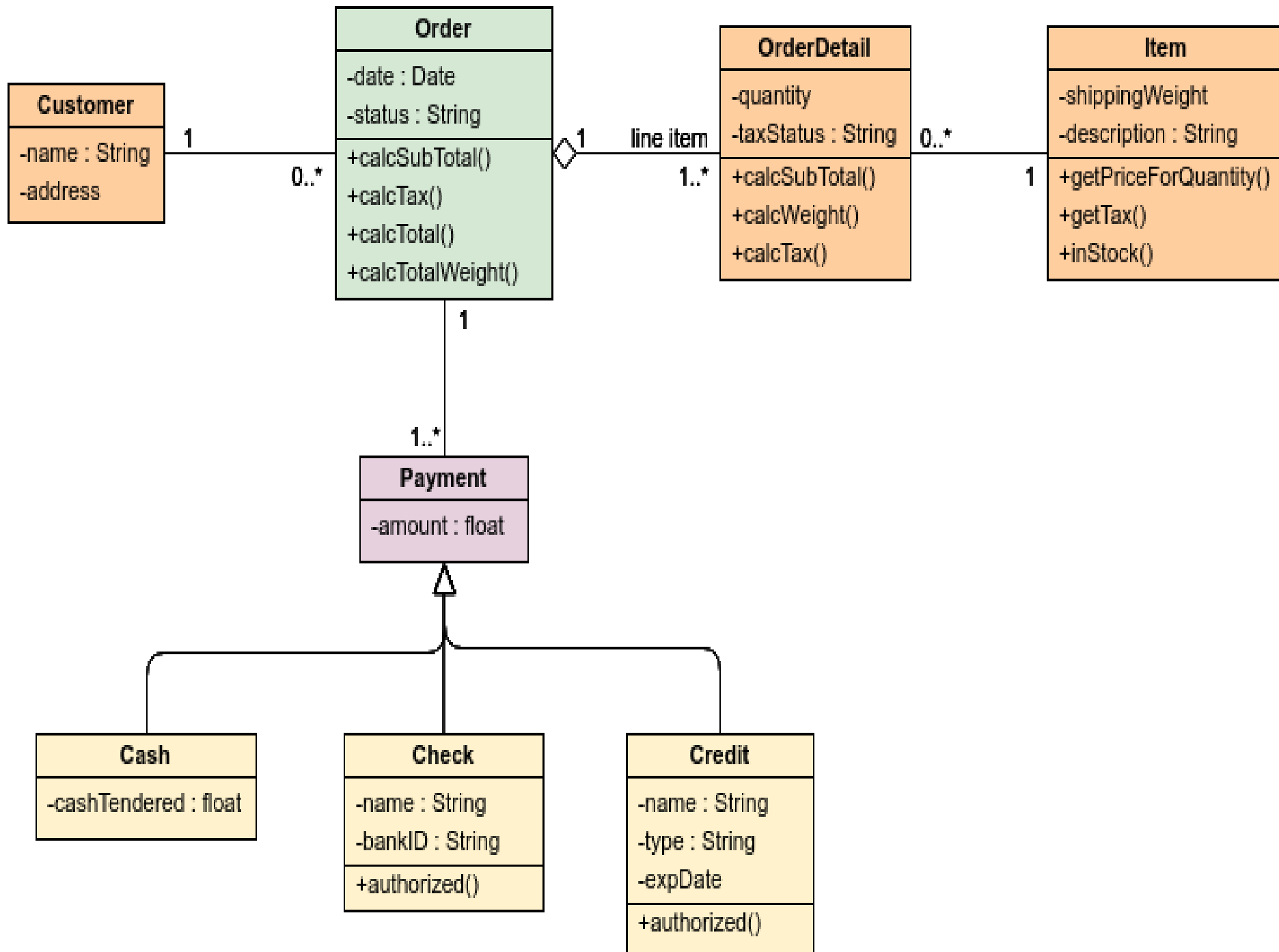


5. Aggregation - It represents a "part of" relationship.
A solid line with an unfilled diamond at the association end connected to the class of composite.



6. Composition - A special type of aggregation where parts are destroyed when the whole is destroyed. A solid line with a filled diamond at the association connected to the class of composite





ABSTRACT CLASS IN CLASS DIAGRAM

- An abstract class name should be written in italics format.
- It is also possible to have an abstract class with no operations declared inside of it.
- In UML, the abstract class has the same notation as that of the class. The only difference between a class and an abstract class is that the class name is strictly written in an italic font.

Rules must be taken care of while representing a class:

- A class name should always start with a capital letter.
- A class name should always be in the center of the first compartment.
- A class name should always be written in **bold** format.
- An abstract class name should be written in **italics** format.

Aggregation vs. Composition

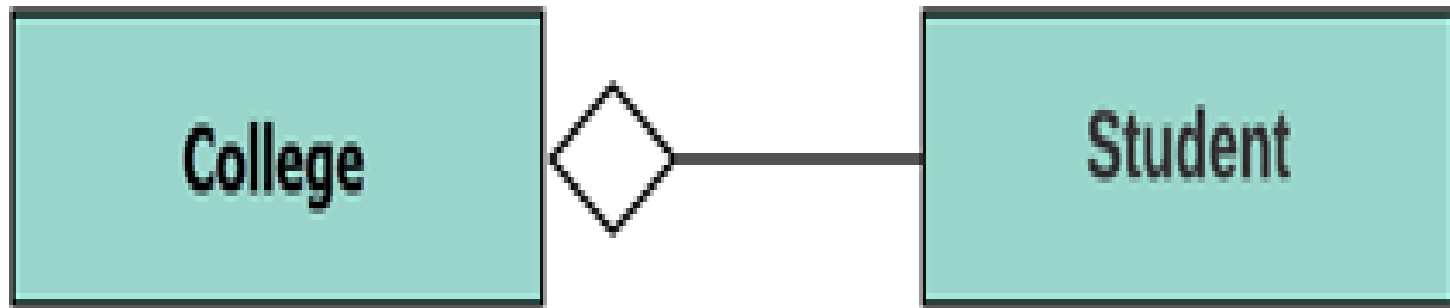
Aggregation

Aggregation indicates a relationship where the child can exist separately from their parent class.
Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist.

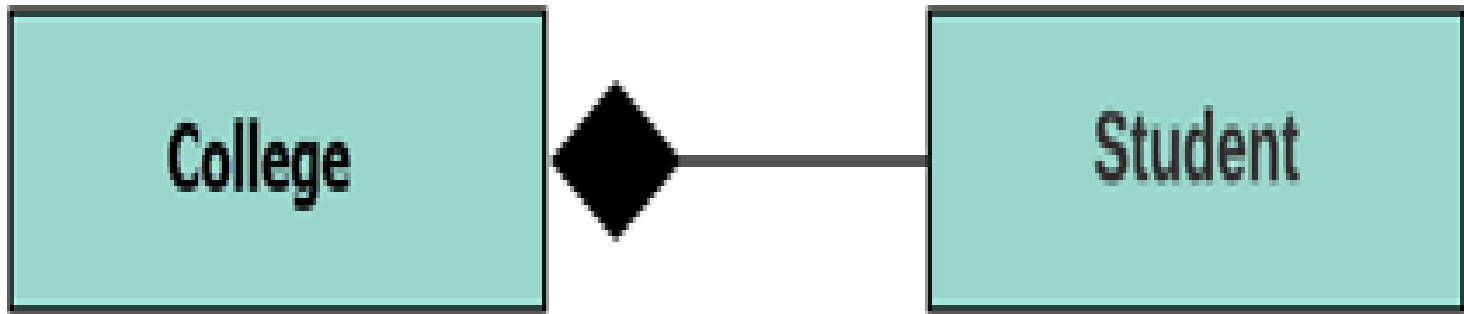
Composition

Composition display relationship where the child will never exist independent of the parent.
Example: House (parent) and Room (child). Rooms will never separate into a House.

Aggregation



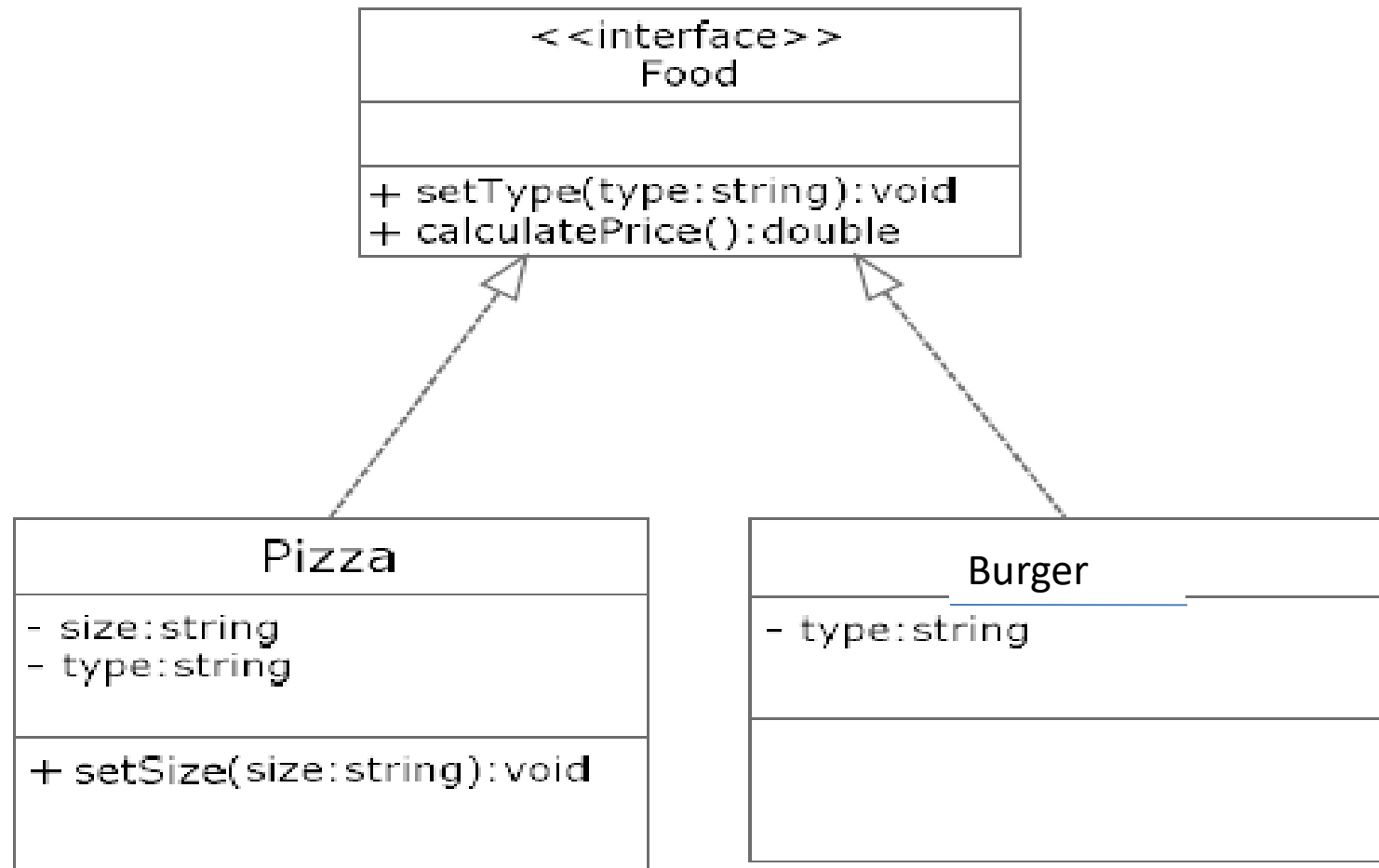
Composition



Interface

- In UML modeling, *interfaces* are model elements that define sets of operations that other model elements, such as classes, or components must implement.
- Each interface specifies a well-defined set of operations that have public visibility.
- Class rectangle symbol that contains the keyword «interface». This notation is also called the internal or class view.

Example of an Interface



Class Diagram in Software Development Lifecycle

- Class diagrams can be used in various software development phases. It helps in modeling class diagrams in three different perspectives.

- 1. Conceptual perspective**
- 2. Specification perspective**
- 3. Implementation perspective:**

GRASP

- **GRASP** stands for general responsibility assignment software principles/Patterns.
- GRASP also helps us define how classes work with one another. The key point of GRASP is to have efficient, clean, understandable code.

GRASP include nine principles

- 1. Creator**
- 2. Information expert**
- 3. Low coupling**
- 4. Controller**
- 5. High cohesion**
- 6. Indirection**
- 7. Polymorphism**
- 8. Protected variations**
- 9. Pure fabrication**

Creator

- In GRASP, we're not worried about the mechanics of object-oriented design.
- The **creator** defines who instantiates what object. In object-oriented design lingo, we need to ask the question of who creates an object A. The solution is that we give class B the role of **instantiating**, or creating an instance of, a class A, if:
 - B contains A
 - B uses most of A's features

Information Expert

The **information expert** pattern states that we need to assign responsibilities to the right expert.

Low Coupling

- Coupling is a measure of how much objects are tied to one another. We can follow the information expert for the lowest level of coupling.

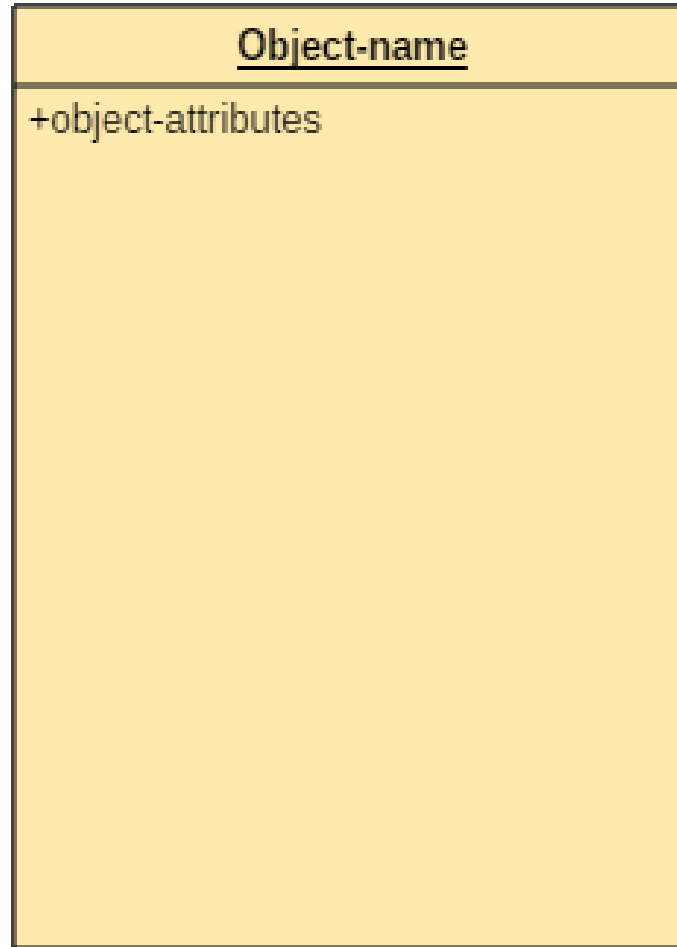
OBJECT DIAGRAM

- Object diagrams gives a representation of a class diagram at any point of time.
- Objects are the real-world entities whose behavior is defined by the classes.
- We cannot define an object without its class. Object and class diagrams are somewhat similar.

OBJECT DIAGRAM

- Class diagram represents the static nature of the system.
- Class come to live only when objects are created from them.
- Object diagram gives the representation of a class with data.

Notation of an object diagram:



How to draw an object diagram?

- Before drawing an object diagram, one should analyze all the objects inside the system.
- The relations of the object must be known before creating the diagram.
- Association between various objects must be cleared before.
- An object should have a meaningful name that describes its functionality.
- An object must be explored to analyze various functionalities of it.

Purpose of an object diagram

- It is used to describe the static aspect of a system.
- It is used to represent an instance of a class.
- It can be used to perform forward and reverse engineering on systems.
- It is used to understand the behavior of an object.
- It can be used to explore the relations of an object and can be used to analyze other connecting objects.

How the class diagram is mapped to the object diagram?

Elements of object diagram

1. Objects

Objects are instances of a class. For example, if "car" is a class.

2. Class titles

Class titles are the specific attributes of a given class. In the family tree object diagram, class titles include the name, gender, and age of the family members. You can list class titles as items on the object or even in the properties of the object itself (such as color).

3. Class attributes

Class attributes are represented by a rectangle with two tabs that indicates a software element.

4. Links

Links are the lines that connect two shapes of an object diagram to each other. The corporate object diagram below shows how departments are connected in the traditional organizational chart style.

CRC CARDS

- **Stands for Class-Responsibility-Collaborator cards**
- Class-responsibility-collaborator cards (CRC cards) are not a part of the UML specification, but they are a useful tool for organising classes during analysis and design.
- A CRC card is a physical card representing a single class. Each card lists the class's name, attributes and methods (its responsibilities), and class associations (collaborations). The collection of these CRC cards is the CRC model.

CRC CARDS

➤ Using CRC cards is a straightforward addition to object-oriented analysis and design:

- Identify the classes.
- List responsibilities.
- List collaborators.

CRC cards can be used during analysis and design while classes are being discovered in order to keep track of them.

Benefits of CRC CARDS

- They are **portable**: because CRC cards are physical objects, they can be used anywhere. Importantly, they can easily be used during group meetings.
- They are **tangible**: the participants in a meeting can all easily examine the cards, and hence examine the system.
- They have a **limited size**: because of their physicality, CRC cards can only hold a limited amount of information. This makes them useful to restricting object-oriented analysis and design from becoming too low-level.

CRC CARDS

- Class responsibilities are the class's attributes and methods. Clearly, they represent the class's state and behaviour. Collaborators represent the associations the class has with other classes.
- CRC cards are useful when the development of classes need to be divided between software engineers, as the cards can be physically handed over to them. A useful time to do this is when classes are being reviewed, for, say, determining whether they are appropriate in a design.
-

CRC CARD LAYOUT

Class Name	
Responsibilities	Collaborators

PACKAGE DIAGRAM

PACKAGE DIAGRAM

- Package diagram is used to simplify complex class diagrams, you can group classes into packages. A package is a collection of logically related UML elements.
- Packages appear as rectangles with small tabs at the top.
- The package name is on the tab or inside the rectangle.

Basic Concepts of Package Diagram

- Package name should not be the same for a system, however classes inside different packages could have the same name.
- Packages can include whole diagrams, name of components alone or no components at all.

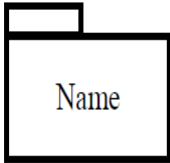
Key Elements of Package Diagram

Packages are used to organize a large set of model elements:

1.Visibility

2.Import

3.Access

Construct	Description	Syntax
Package	A grouping of model elements.	
Import	A dependency indicating that the public contents of the target package are added to the namespace of the source package.	«import» ----->
Access	A dependency indicating that the public contents of the target package are available in the namespace of the source package.	«access» ----->

When to Use Packages?

- To create an overview of a large set of model elements
- To organize a large model
- To group related elements
- To separate namespaces

VISIBILITY OF PACKAGE

- A **public** element is visible to elements outside the package, denoted by '+'
- A **protected** element is visible only to elements within inheriting packages, denoted by '#'
- A **private** element is not visible at all to elements outside the package, denoted by '-'
- Same syntax for visibility of attributes and operations in classes

PACKAGE DIAGRAM

- The types of dependencies defined between packages:
- package import
- package merge
- uses

import

- A *package import* is "a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace"

merge

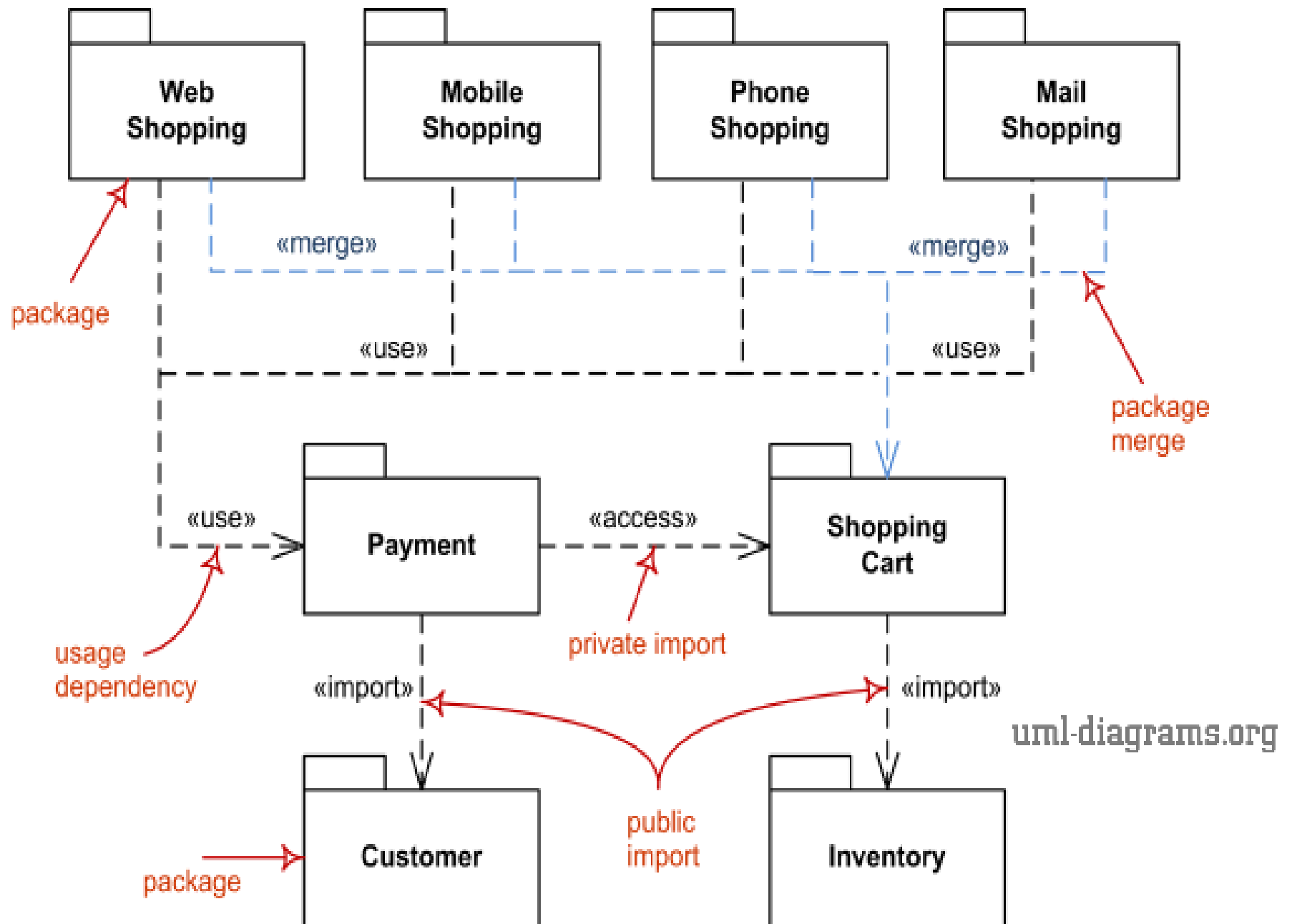
- A *package merge* is "a directed relationship between two packages, that indicates that the contents of the two packages are to be combined.

uses

- the use dependency indicates that a model element - not necessarily a package - requires another model element for its implementation

Uses Vs import

- In use you just pick certain parts from a package while the import takes all.



Uses of Package Diagram

- Package diagrams can use packages containing use cases to illustrate the functionality of a software system.
- Package diagrams can use packages that represent the different layers of a software system to illustrate the layered architecture of a software system.
- The dependencies between these packages can be represented with labels indicate the communication mechanism between the layers.

STATE CHART DIAGRAM

STATE DIAGRAM

- or State Chart
- or State Machine
- or State Transition Diagram

State: Status of a System/Subsystem/Object **at a particular time**

3 Important elements of State Diagram:

- ☐ State
 - ☐ Event/Trigger
 - ☐ Transition
-

STATE CHART DIAGRAM

STATE DIAGRAM

- Shows different states of an System/Subsystem/Object during its life time.
- Used to model Dynamic aspects of a system.
- Its specific purpose is to define the state changes triggered by events.
- Events are internal or external factors influencing the system.

STATE CHART DIAGRAM

- State chart diagram describes the flow of control from one state to another state.
- States are defined as a condition in which an object exists and it changes when some event is triggered.
- The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

STATE CHART DIAGRAM

- Statechart diagram is used to capture the dynamic aspect of a system.
- State machine diagrams are used to represent the behavior of an application.
- An object goes through various states during its lifespan.
- The lifespan of an object remains until the program is terminated.
- The object goes from multiple states depending upon the event that occurs within the object. Each state represents some unique information about the object.

The main purposes of using State chart diagrams

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

Notation and Symbol for State Machine



- **Initial state**

The initial state symbol is used to indicate the beginning of a state machine diagram.

- **Final state**

This symbol is used to indicate the end of a state machine diagram.

- **Decision box**

It contains a condition. Depending upon the result of an evaluated guard condition, a new path is taken for program execution.

- **Transition**

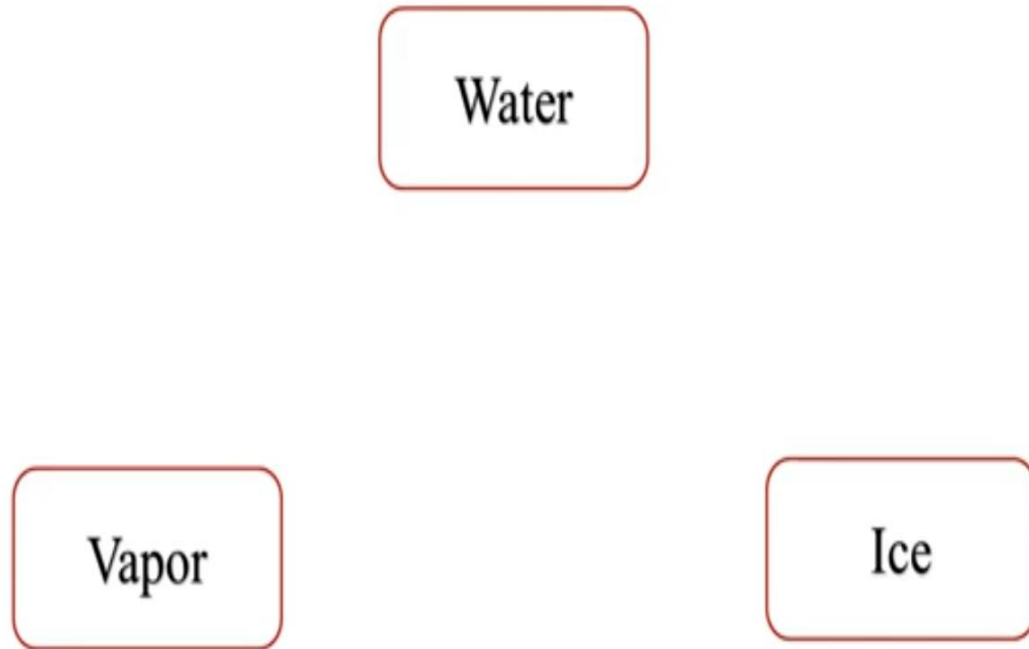
A transition is a change in one state into another state which is occurred because of some event. A transition causes a change in the state of an object.

- **State box**

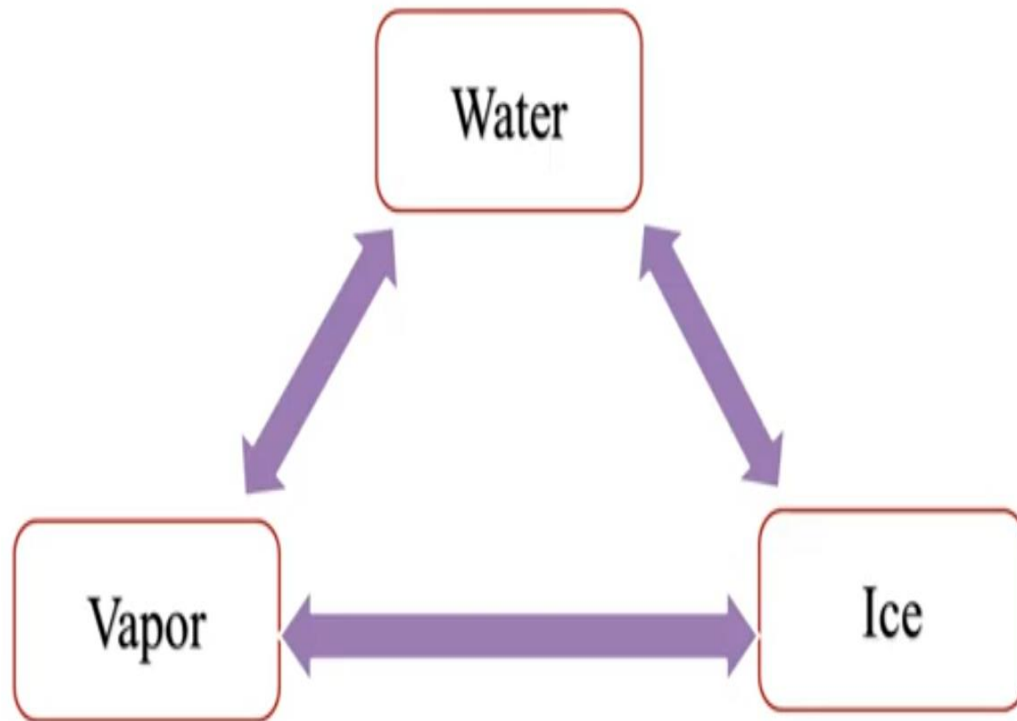
It is a specific moment in the lifespan of an object. It is defined using some condition or a statement within the classifier body. It is used to represent any static as well as dynamic situations.

- It is denoted using a rectangle with round corners. The name of a state is written inside the rounded rectangle.

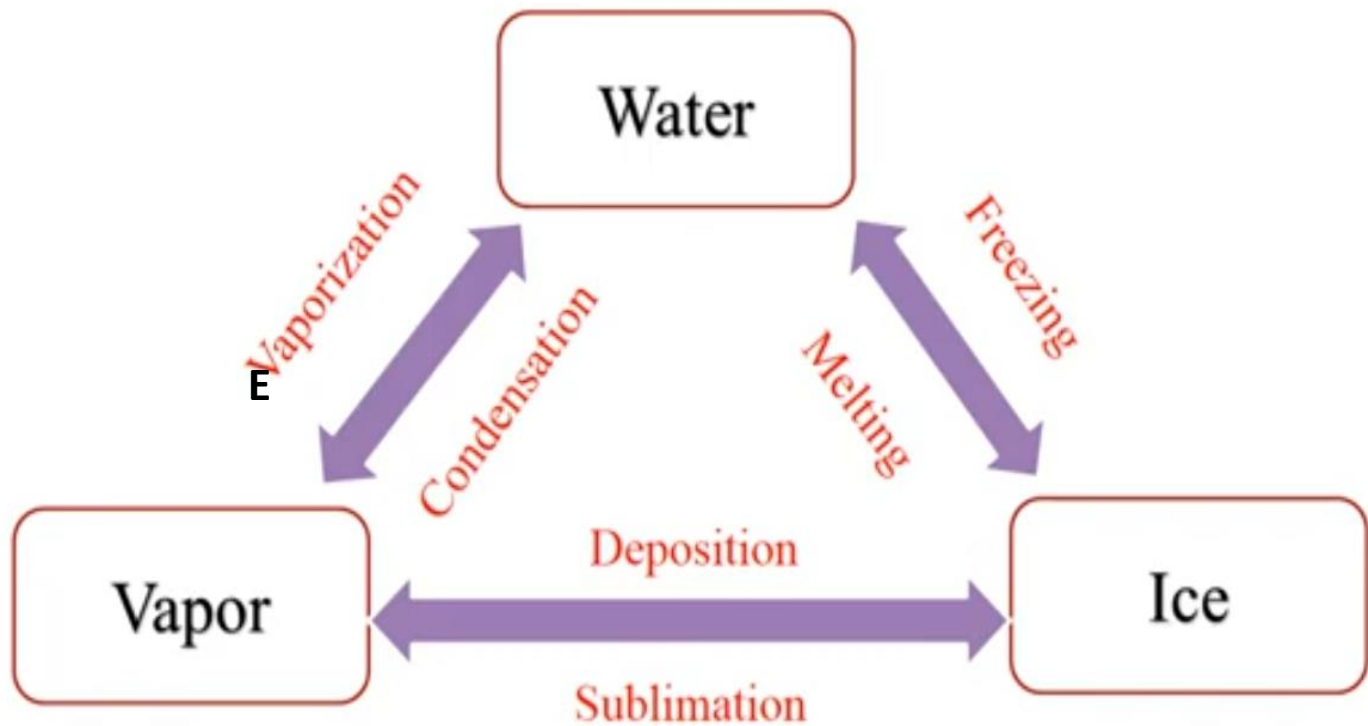
SIMPLE STATE DIAGRAM



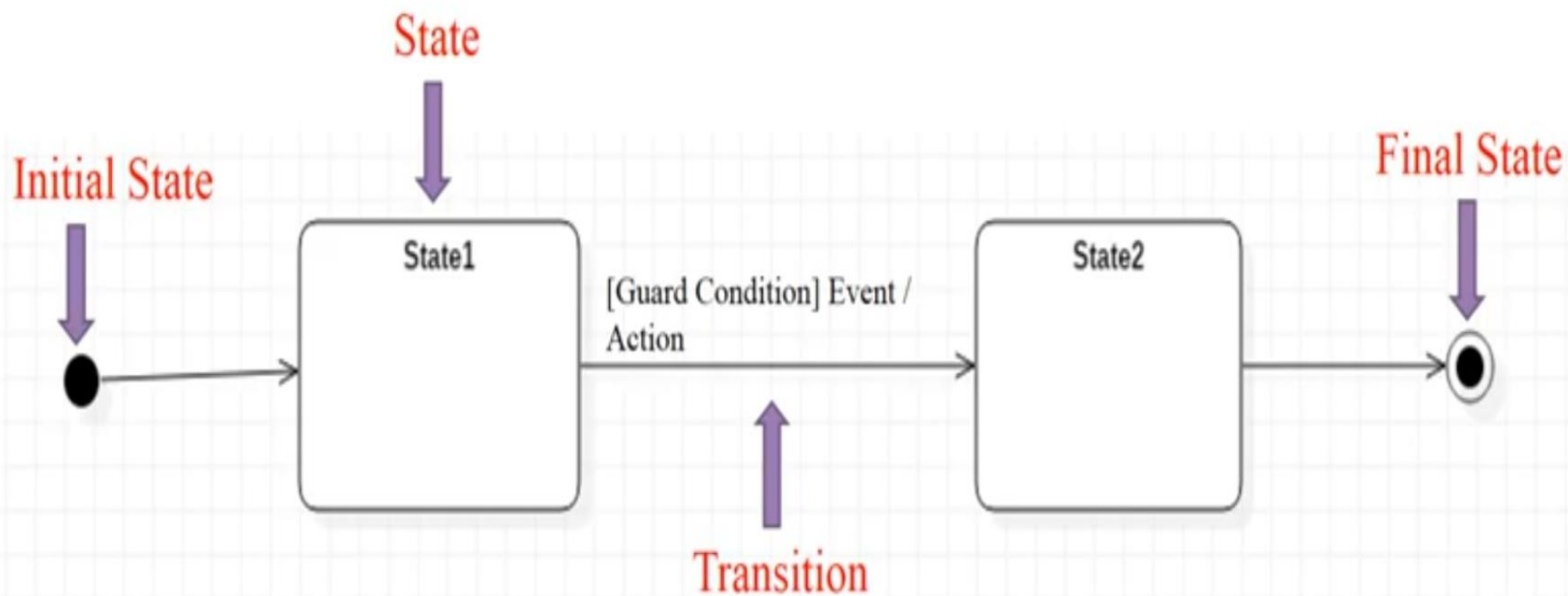
SIMPLE STATE DIAGRAM



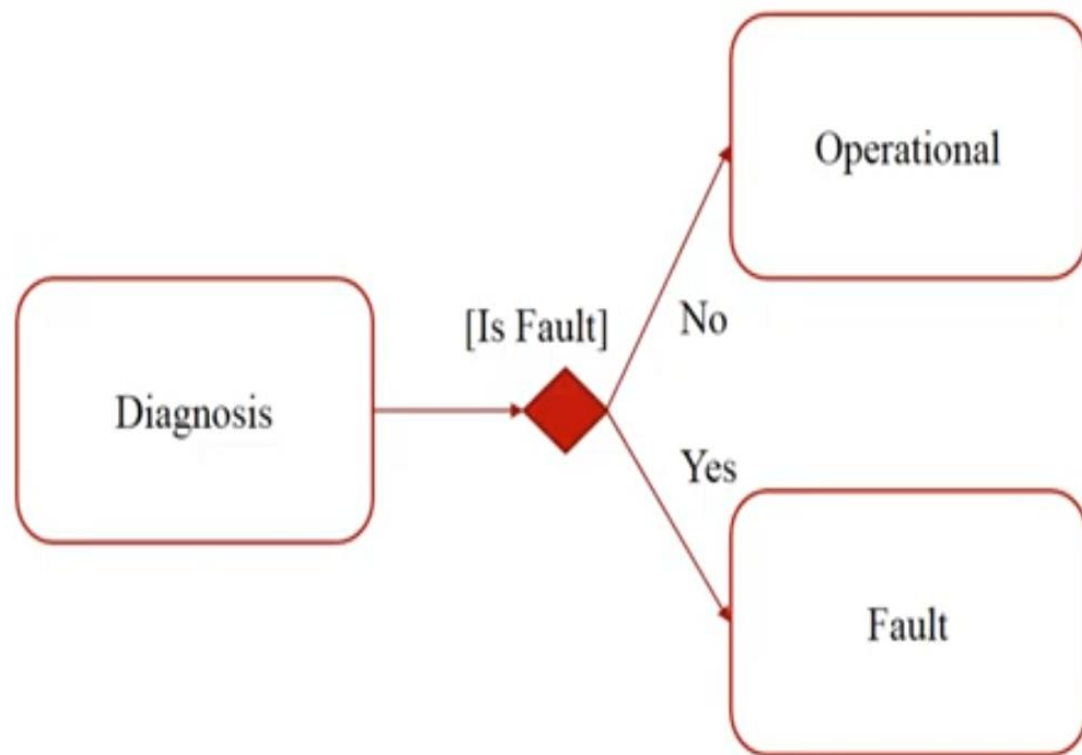
SIMPLE STATE DIAGRAM



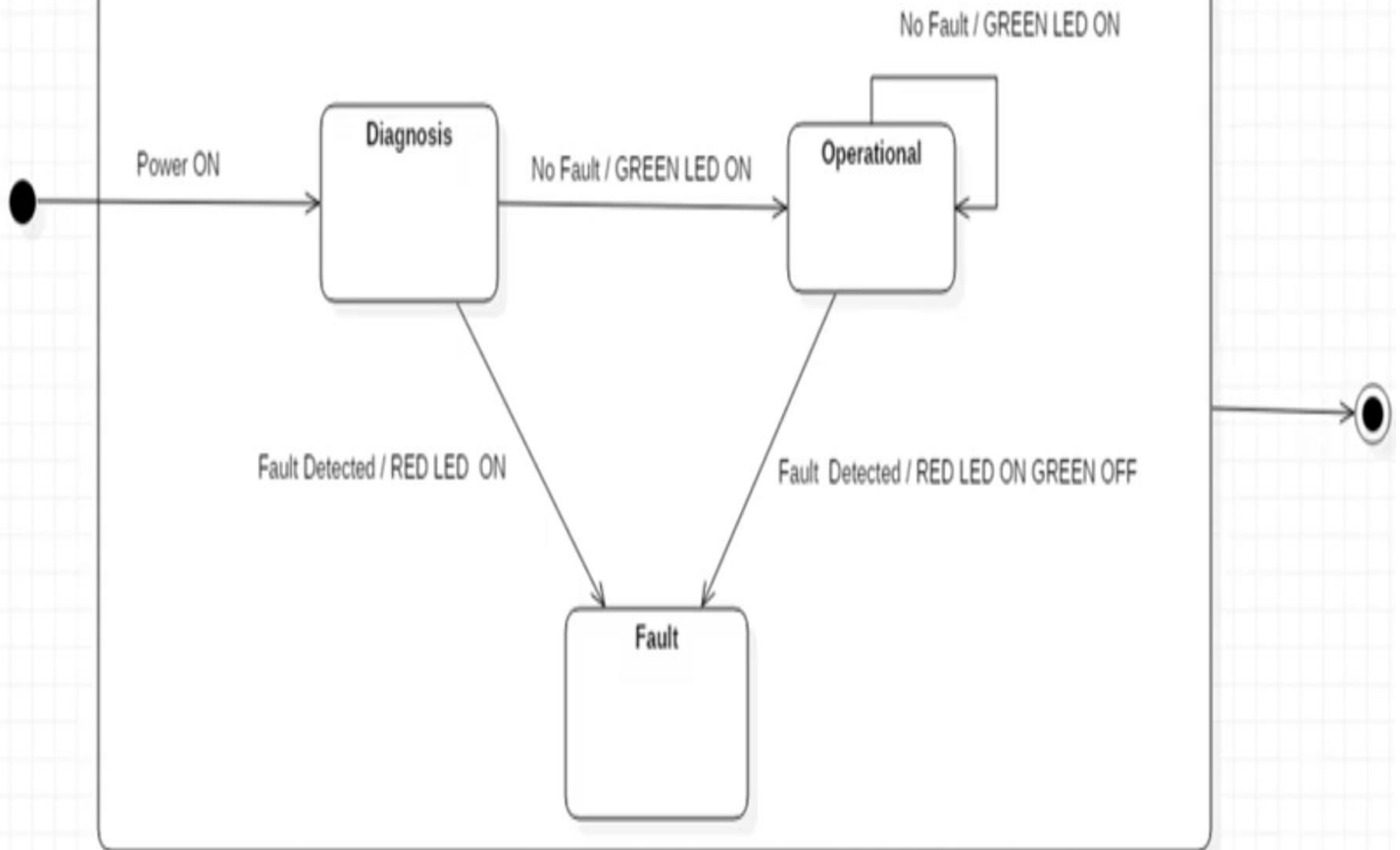
NOTATIONS



EXAMPLE

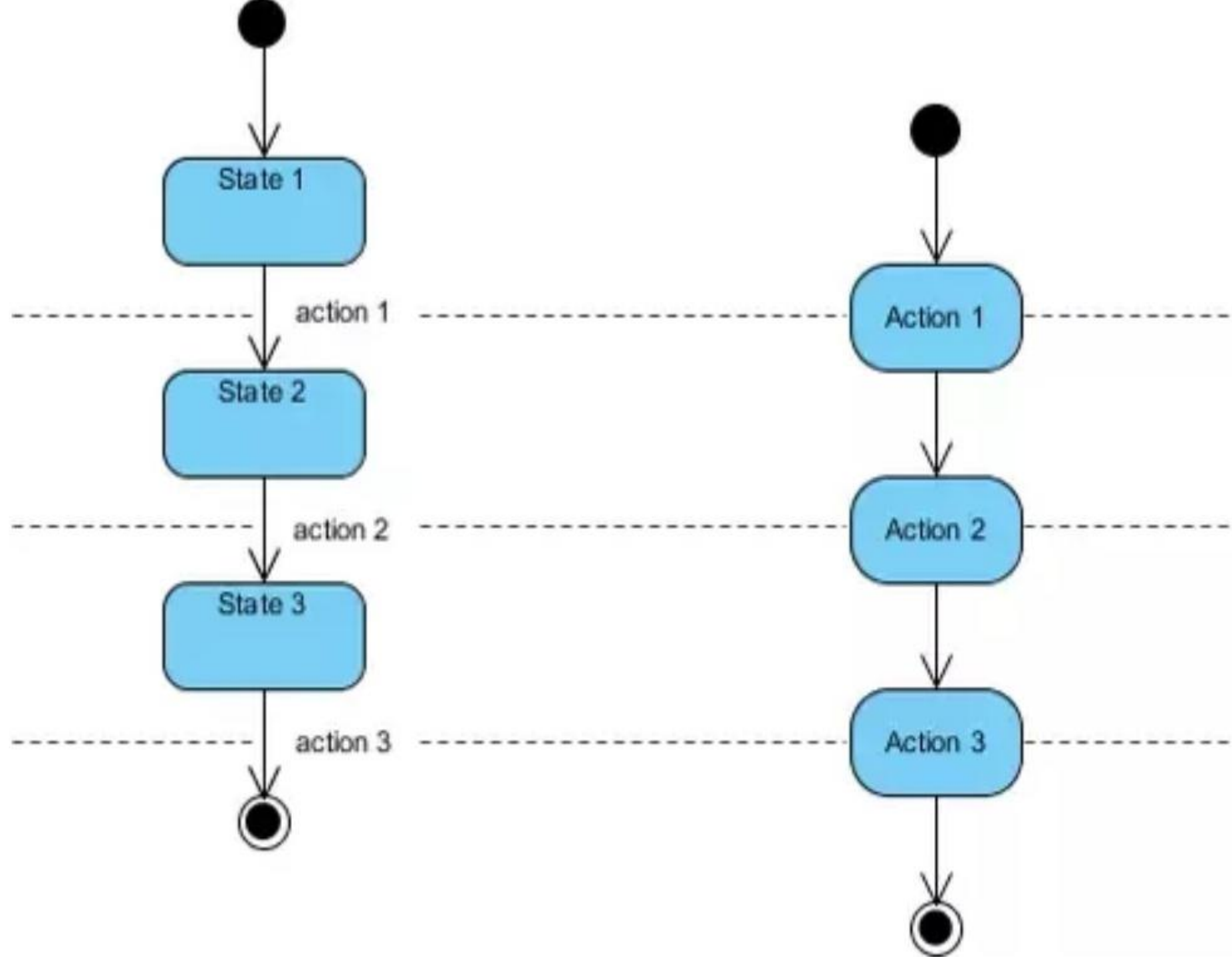


System



STATE CHART Vs ACTIVITY DIAGRAMS

- **activity diagrams** describe activities and **state charts** describe states.
- Activity diagram is used to document the logic of a single operation/method, a single use case or the flow of logic of a business process.
- The state diagram depict (show) the state of objects as their attributes change from state to the other state.
- **Activity diagram** is flow of functions without trigger (event) mechanism, **state machine** is consist of triggered states.



State diagram applications

- Depicting event-driven objects in a reactive system.
- Illustrating use case scenarios in a business context.
- Describing how an object moves through various states within its lifetime.
- Showing the overall behavior of a state machine or the behavior of a related set of state machines.

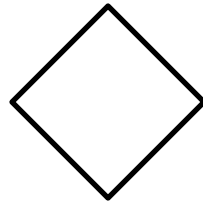
State diagram symbols and components

Composite state

- A state that has substates nested into it

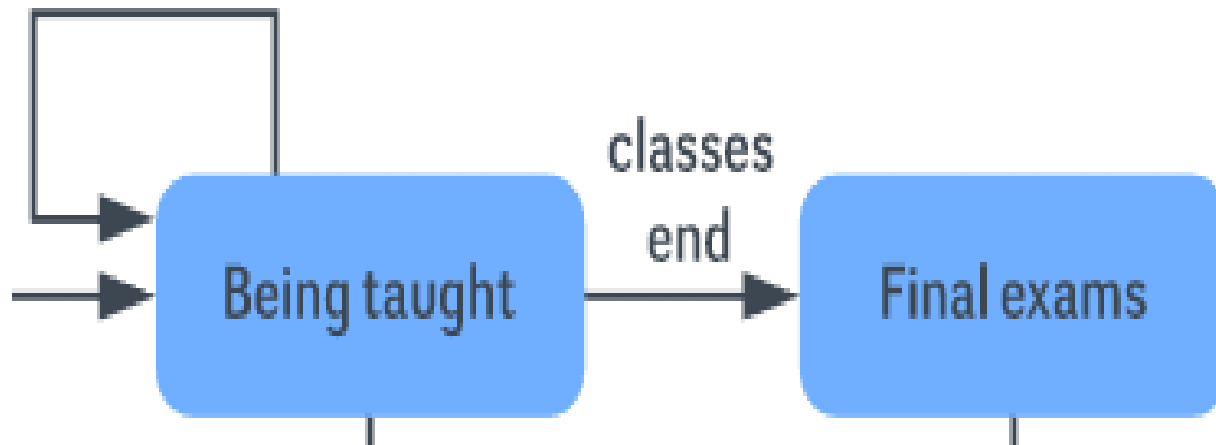
Choice pseudostate

A diamond symbol that indicates a dynamic condition with branched potential results.



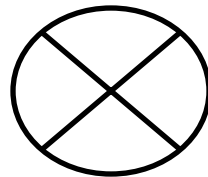
Event

An instance that triggers a transition, labeled above the applicable transition arrow. In this case, “classes end” is the event that triggers the end of the “Being taught” state and the beginning of the “Final exams” state.



Exit point

The point at which an object escapes the composite state or state machine, denoted by a circle with an X through it. The exit point is typically used if the process is not completed but has to be escaped for some error or other issue.



Guard

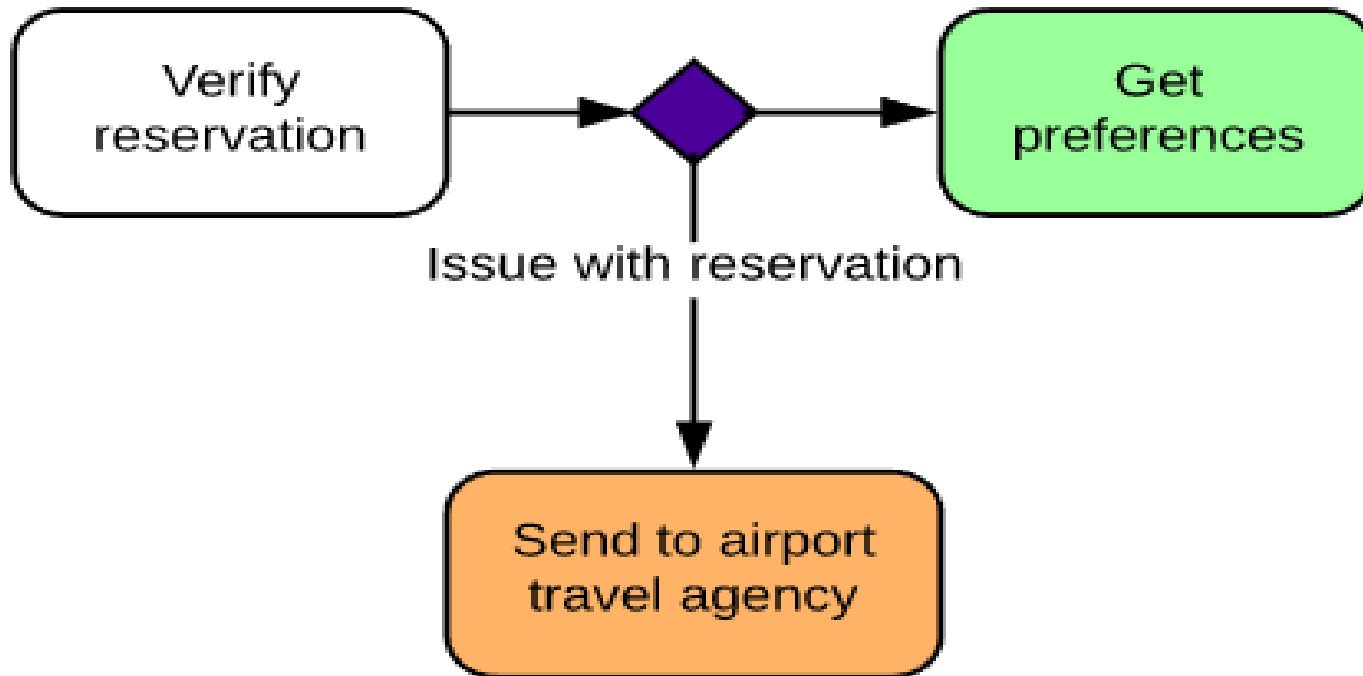
A Boolean condition that allows or stops a transition, written above the transition arrow.

Types of states

- **Composite state** – state that contains states.
- **Substate** - A state contained within a composite state's region.
- **Simple state**
- **Simple state with region containing actions or methods**

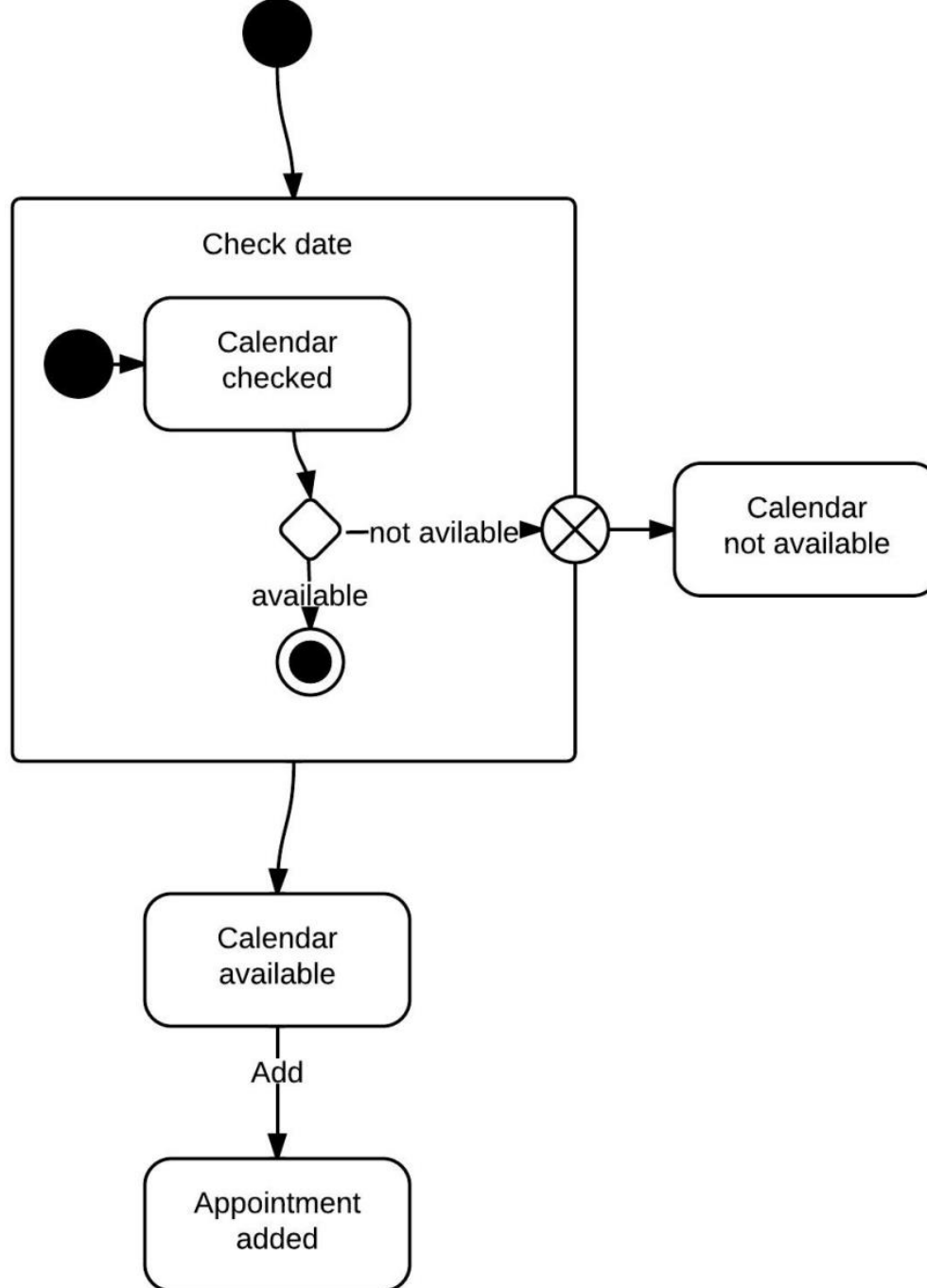
Trigger

A type of message that actively moves an object from state to state, written above the transition arrow. In this example, “Issue with reservation” is the trigger that would send the person to the airport travel agency instead of the next step in the process.



Calendar availability state diagram

example



University state diagram example

