MÔN HỌC:

CÔNG CỤ THIẾT KẾ HỆ THỐNG THÔNG TIN

MÃ MH: **ITEC3407**

SỐ TC: 3 **(2LT, 1TH)**

KHOA CNTT, ĐH MỞ TP. HCM

CHƯƠNG 5: MÔ HÌNH HƯỚNG ĐỐI TƯỢNG

- 5.1. Cơ bản về mô hình HƯỚNG ĐỐI TƯỢNG
- 5.2. Xây dựng biểu đồ CA SỬ DỤNG, LỚP, ĐỐI TƯỢNG
- 5.3. Xây dựng biểu đồ HỢP TÁC, TUẦN TỰ,
- 5.4. Xây dựng biểu đồ TRẠNG THÁI, HOẠT ĐỘNG, THÀNH PHẦN, TRIỂN KHAI
- 5. Làm việc với mô hình HƯỚNG ĐỐI TƯỢNG
- 5.6. Các mô hình được sinh ra từ MH HƯỚNG ĐT
- 5.7. Tổng kết chương & Bài tập

5.1. CƠ BẢN VỀ MÔ HÌNH HƯỚNG ĐỐI TƯỢNG

5.1.1. Tổng quan về chức năng

5.1.2. Mô hình UML và hướng đối tượng

5.1.3. OOM (Object Oriented Model) là gì?

5.1.4. Định nghĩa OOM

5.1.5. Định nghĩa các tùy chọn cho OOM

5.1.6. Định nghĩa các gói (package) trong OOM

5.1.1. TỔNG QUAN VỀ CHỨC NĂNG

Functional overview

PowerDesigner Object-Oriented Model is a powerful design tool for graphical object-oriented design implementation.

With this product, you can:

- Build an Object-Oriented Model (OOM) using the **use case**, **class, object, collaboration, sequence**, **statechart, activity**, **component** or **deployment** diagrams
- Generate Java, XML files, Sybase PowerBuilder objects
- Generate code for C++, Visual Basic, C#, IDL-CORBA, etc.
- Generate EJB, servlets, JSP, ASP.NET components
- Generate a Conceptual Data Model (CDM),a Physical Data Model (PDM) and an Object-Oriented Model from the OOM
- Reverse engineer Java, XML files, PowerBuilder objects
- Import a Conceptual Data Model (CDM)
- Import a Physical Data Model (PDM)
- Import a Rose model
- Import/export XMI
- Customize the OOM to suit physical and performance considerations
- Customize and print model reports

5.1.2. MÔ HÌNH UML VÀ HƯỚNG ĐỐI TƯỢNG

UML and object-oriented modeling

What is UML?

UML (Unified Modeling Language) is a modeling language aimed at defining standards for object-oriented modeling. UML has become a standardized language largely through the work of the **OMG** (Object Management Group), a group composed of individuals and representatives of companies involved in object-oriented projects. However, its original conception drew much of its inspiration from the work of Grady Booch, James Rumbaugh, and Ivar Jacobson.

UML has a vocabulary and rules that focus on the conceptual and physical representation of a system. You use UML symbols and notations to create your models and diagrams.

5.1.2. MÔ HÌNH UML VÀ HƯỚNG ĐỐI TƯỢNG

Notational Terminology	UML has a well-defined syntax and semantics that is clear and easy to use in object modeling. The terminology used in the OOM interface is consistent with UML language notations.

What is object-oriented modeling?	Object-oriented modeling refers to the process of using objects as the basic building blocks for creating a software system. An object in this context usually means a class, that is, a description of a set of common objects. Each object or class has identity and behavior. They communicate with each other through different types of relationships which have their own symbols. You use these objects to build models in which the properties of each object interact to perform certain actions that together make up a system of information.

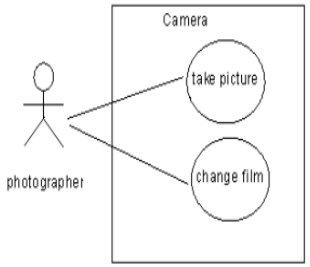
5.1.3. OOM (OBJECT ORIENTED MODEL) LÀ GÌ? (1)

What is an OOM?

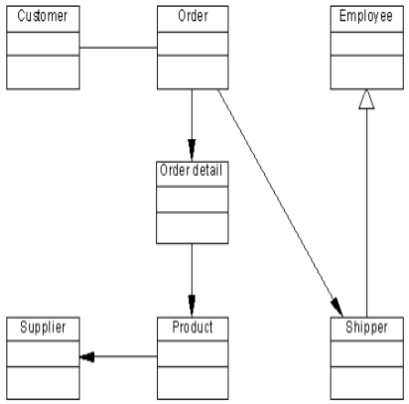
You use an Object-Oriented Model to build an OOM. An OOM is a structure which provides a close description of a system using the following UML diagrams:

5.1.3. OOM (OBJECT ORIENTED MODEL) LÀ GÌ? (2)

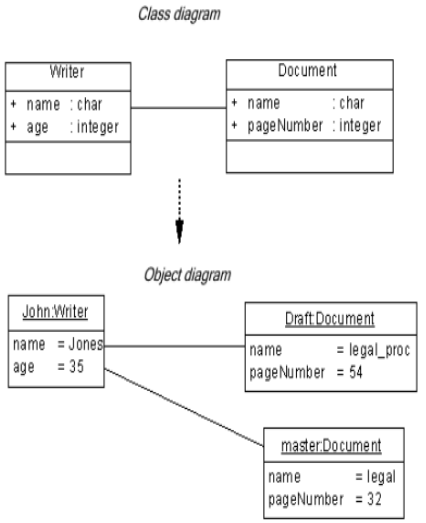
♦ **Use case diagrams**



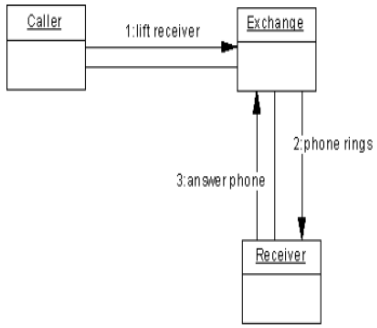
♦ **Class diagrams**



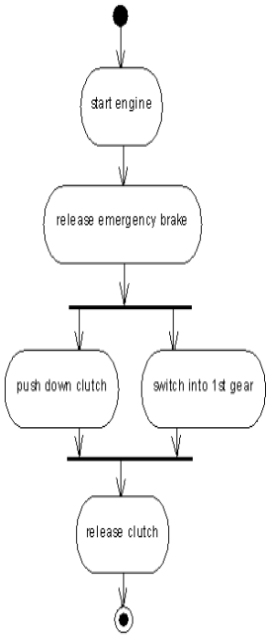
♦ **Object diagrams**



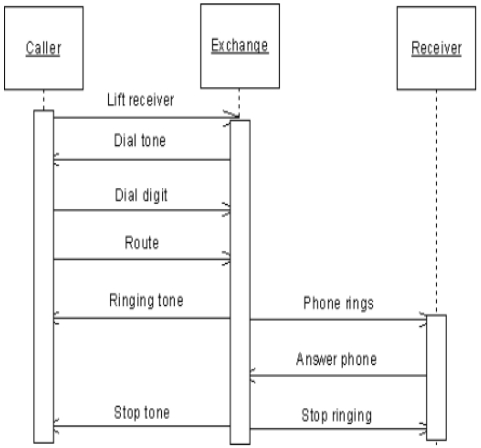
♦ **Collaboration diagrams**



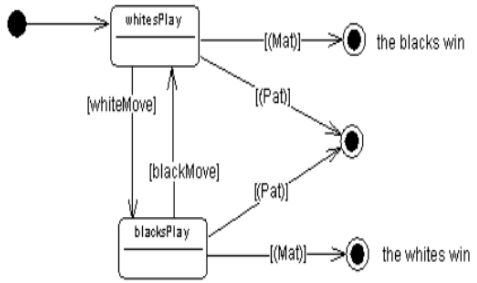
♦ **Activity diagrams**



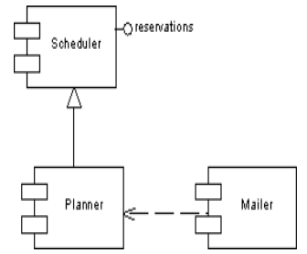
♦ **Sequence diagrams**



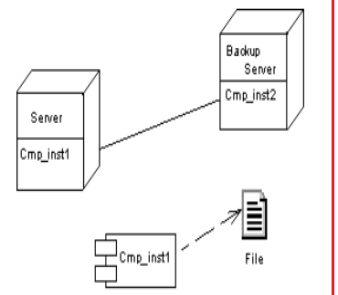
♦ **Statechart diagrams**



♦ **Component diagrams**



♦ **Deployment diagrams**



5.1.3. OOM (OBJECT ORIENTED MODEL) LÀ GÌ? (3)

What is an OOM?

You use an Object-Oriented Model to build an OOM. An OOM is a structure which provides a close description of a system using the following UML diagrams:

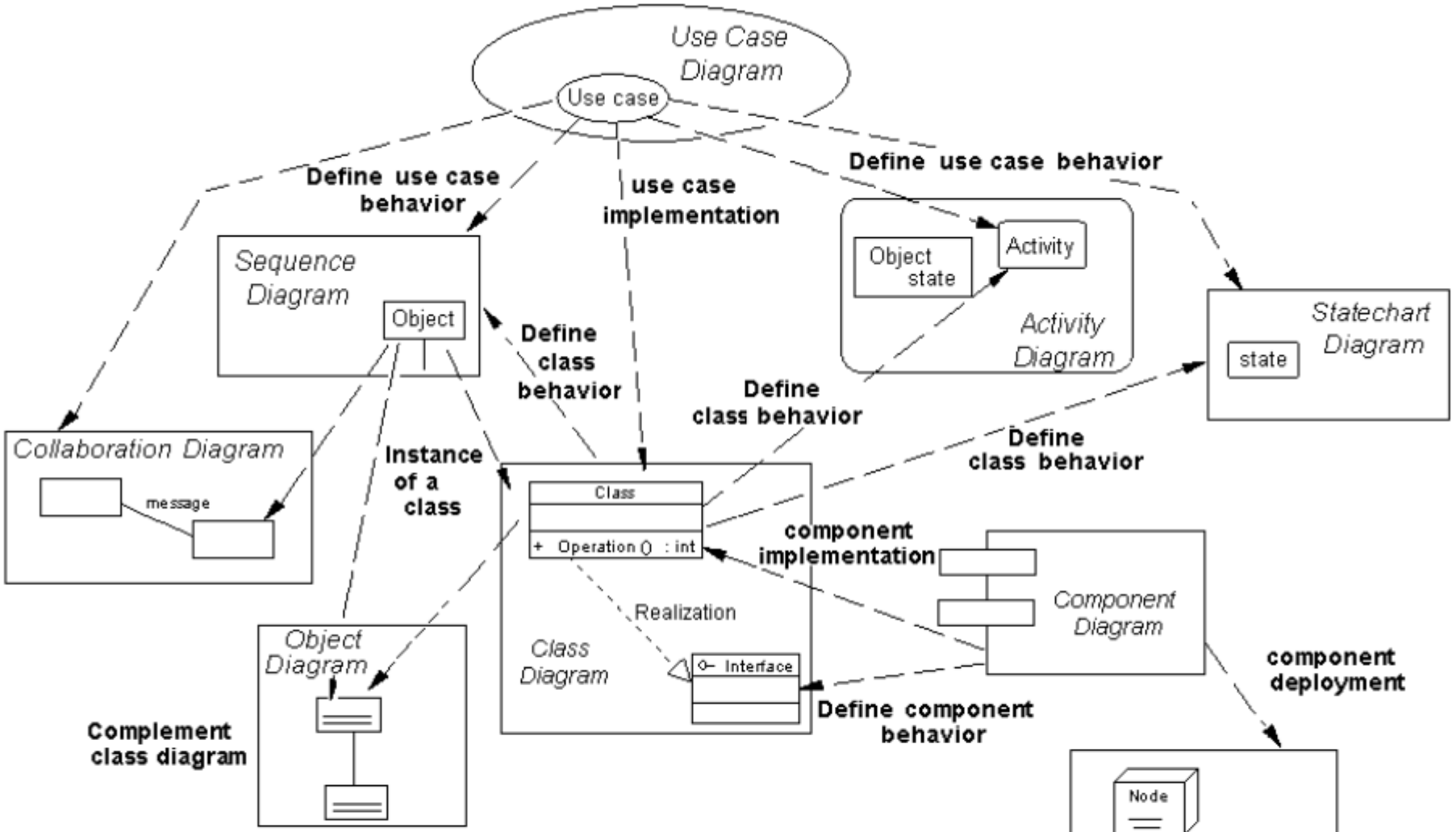
- **Use case diagrams** define the fundamental structure of your application. It allows you to draw a high-level sketch of the system: you define who are the external actors of the system and what activities they perform using it.
- **Class diagrams** define the static structure of the model. The class diagram contains a set of packages, classes, interfaces, and their relationships.
- **Object diagrams** describe the structure of model elements. They complement class diagrams, and display objects (instances of class), instance links (instances of association), and dependencies.
- **Collaboration diagrams** describe how the system achieves what was described in a use case diagram. They convey the same information as sequence diagrams, the main difference is that collaboration diagrams focus on objects in action, they show a network of objects that are

5.1.3. OOM (OBJECT ORIENTED MODEL) LÀ GÌ? (4)

What is an OOM?

- **Sequence diagrams** describe how the system achieves what was described in a use case diagram. You use a sequence diagram to draw objects (instances of a class), and show a time-ordered series of sequenced method invocations among objects. The following example
- **Statechart diagrams** describe the public behavior of a unique classifier (use case, component or class).
- **Activity diagrams** model the dynamic aspects of a system. They describe the flows driven by internal processing from a start point to several potential end points.
- **Component diagrams** show the dependencies among software components, including source code components, binary code components, and executable components.
- **Deployment diagrams** are implementation diagrams that complement component diagrams by giving more accurate details about the physical implementation and interactions of components.

5.1.3. OOM (OBJECT ORIENTED MODEL) LÀ GÌ? (5)



In the picture below, you can see how use case, class, object, collaboration, sequence, statechart, activity, component, and deployment diagrams can interact within the same model:

5.1.4. ĐỊNH NGHĨA OOM

Defining an OOM

You can create a new OOM, open an existing one, close or detach an OOM from the workspace. An OOM is attached to one object language that must be selected at creation.

Creating an OOM

There are several ways to create an OOM:

- Create a new OOM
- Import one or more existing OOM
- Generate an OOM from a CDM
- Generate an OOM from PDM
- Import a Rational Rose model (.mdl)
- Reverse engineer an object language into an OOM

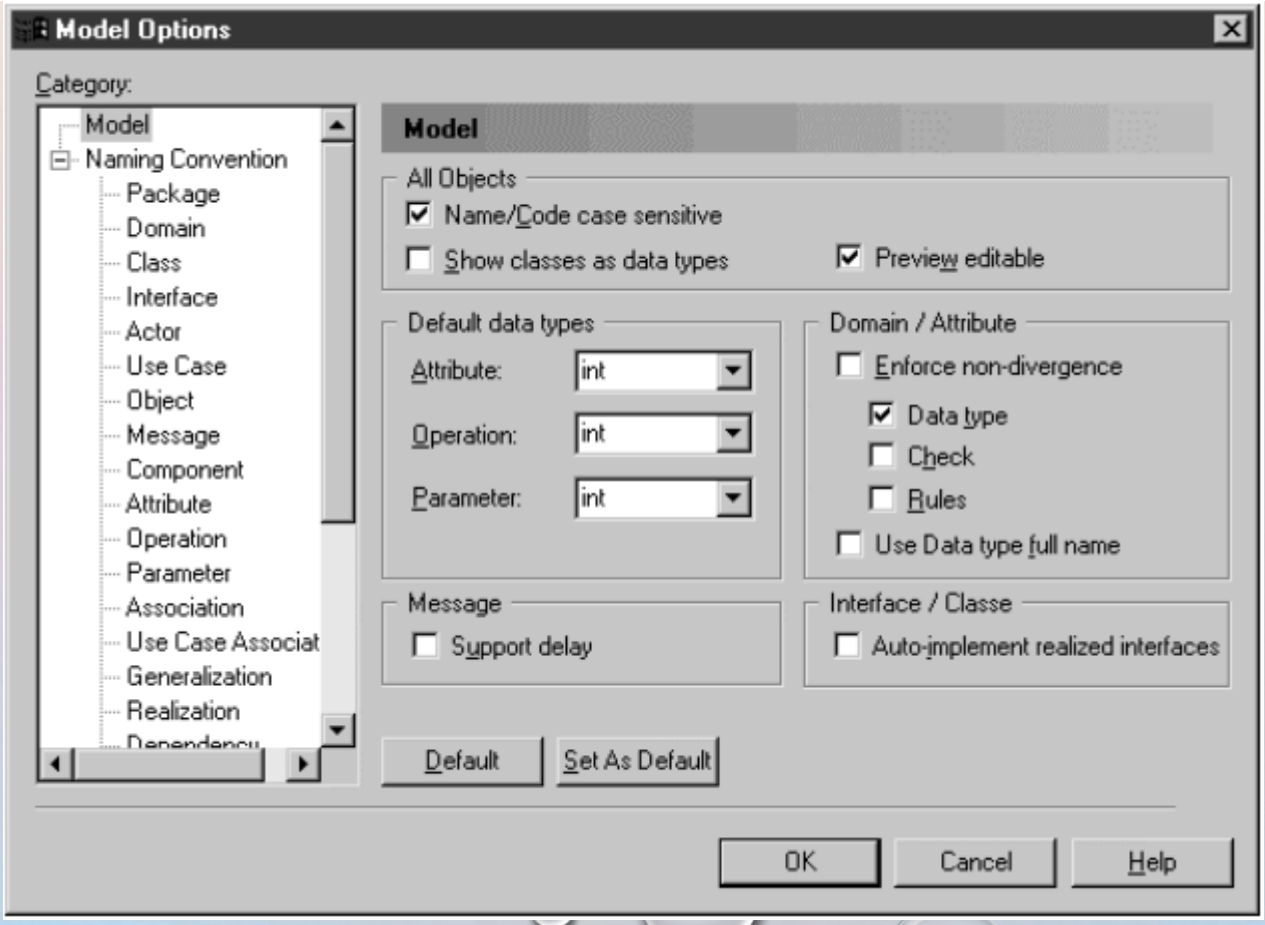
When you create a new OOM, you have to define creation parameters.

5.1.5. ĐỊNH NGHĨA CÁC TÙY CHỌN CHO OOM

Defining OOM options

You can set options to define how your model is created or modified, and the type of information that it displays. Those options apply to the model and to all its objects.

You can also set naming conventions for each type of objects in your model.

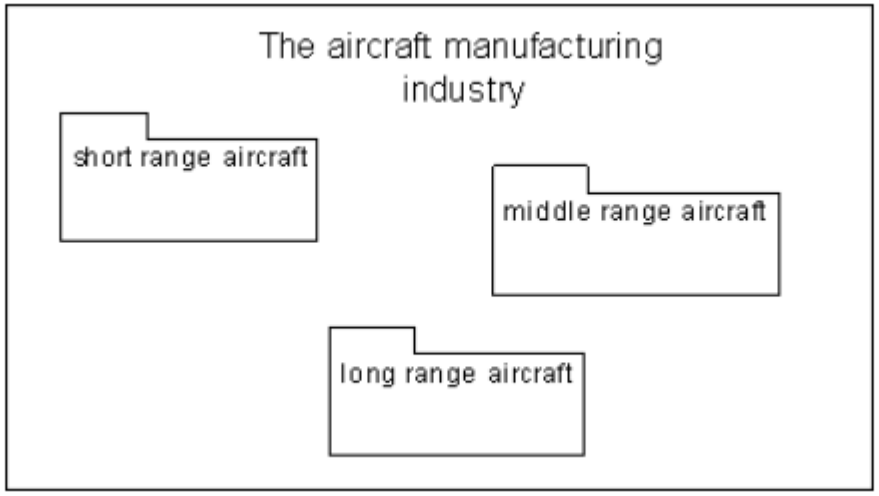


5.1.6. ĐỊNH NGHĨA CÁC GÓI (PACKAGE) TRONG OOM

Defining packages in an OOM

A **package** is a general purpose mechanism for organizing elements into groups. It contains model objects and is available for creation in all diagrams.

When you work with large models, you can split them into smaller subdivisions to avoid manipulating the entire set of data of the model. Packages can be useful to assign portions of a model, representing different tasks and subject areas to different development teams.



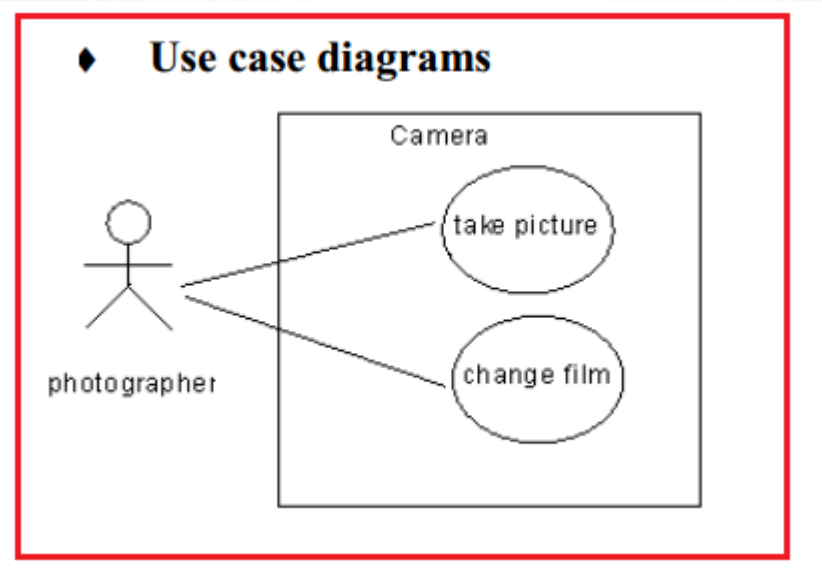
5.2. XÂY DỰNG BIỂU ĐỒ CA SỬ DỤNG, LỚP, ĐỐI TƯỢNG

5.2A. Xây dựng biểu đồ CA SỬ DỤNG

5.2B. Xây dựng biểu đồ LỚP

5.2C. Xây dựng biểu đồ ĐỐI TƯỢNG

5.2A. XÂY DỰNG BIỂU ĐỒ CA SỬ DỤNG



5.2.1. Cơ bản về biểu đồ CA SỬ DỤNG (b.đồ CA)

5.2.2. Định nghĩa các Use Case trong biểu đồ CA

5.2.3. Định nghĩa các Actor (tác nhân) trong b.đ CA

5.2.4. Định nghĩa các Association (liên kết) trong b.đồ CA

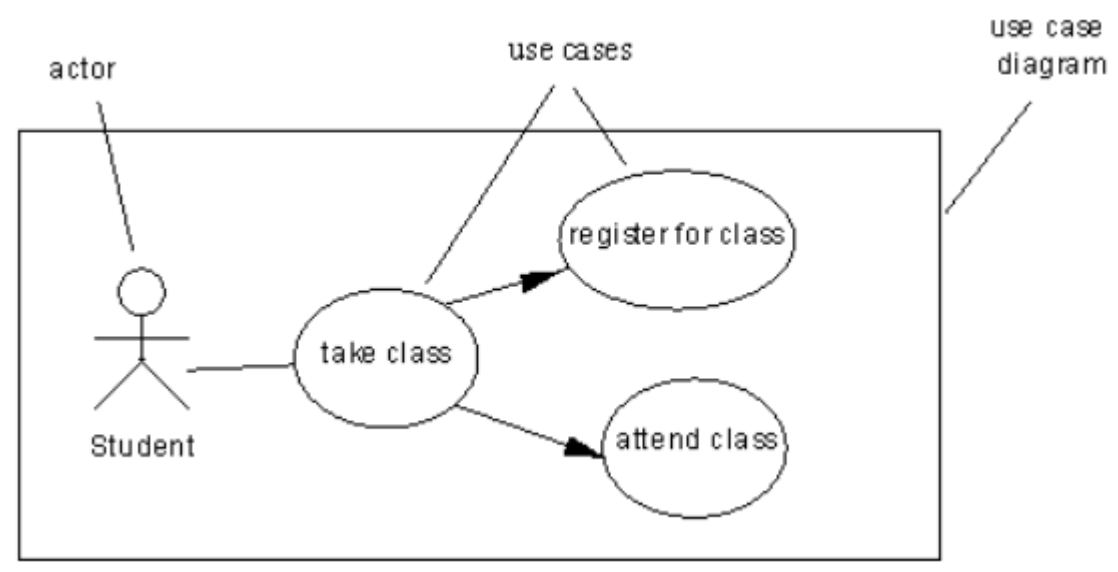
5.2.5. Định nghĩa Generalization trong b.đồ CA

5.2.6. Định nghĩa Dependency (phụ thuộc) trong b.đồ CA

5.2A.1. CƠ BẢN VỀ BIỂU ĐỒ CA SỬ DỤNG (B.ĐỒ CA)

Use case diagram basics

A **use case diagram** is a diagram used to define the behavior of a system. It shows interactions between actors (the external users) and use cases (the services performed in the system).



5.2A.2. ĐỊNH NGHĨA CÁC USE CASE TRONG BIỂU ĐỒ CA

Defining use cases in a use case diagram

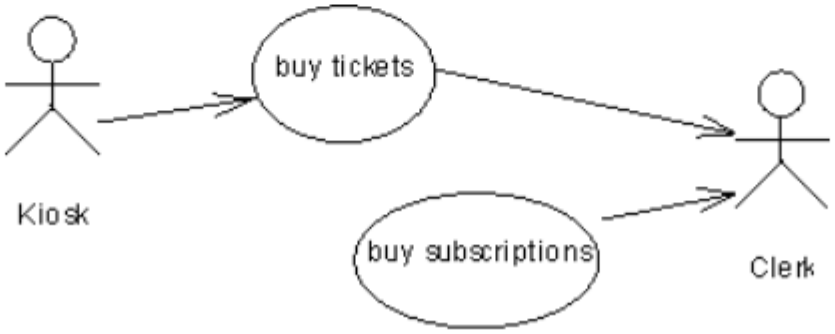
A **use case** could be defined as a service that a system performs.

A use case defines a piece of behavior of a classifier (a system, a sub-system, or a class) as it appears to an external user, without revealing the internal structure of the classifier.

Its purpose is to specify a sequence of actions that a system, or sub-system can perform, interacting with actors of the system.

Example

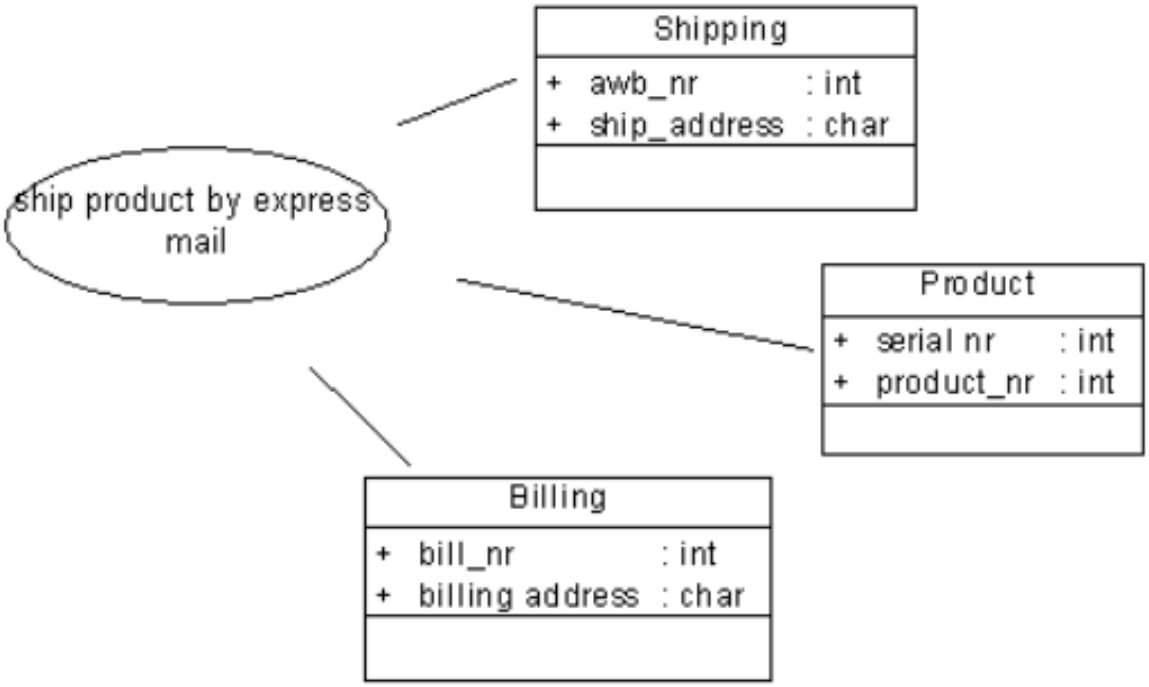
In our example, buy tickets and buy subscriptions are use cases.



5.2A.2. ĐỊNH NGHĨA CÁC USE CASE TRONG BIỂU ĐỒ CA

- VÍ DỤ:

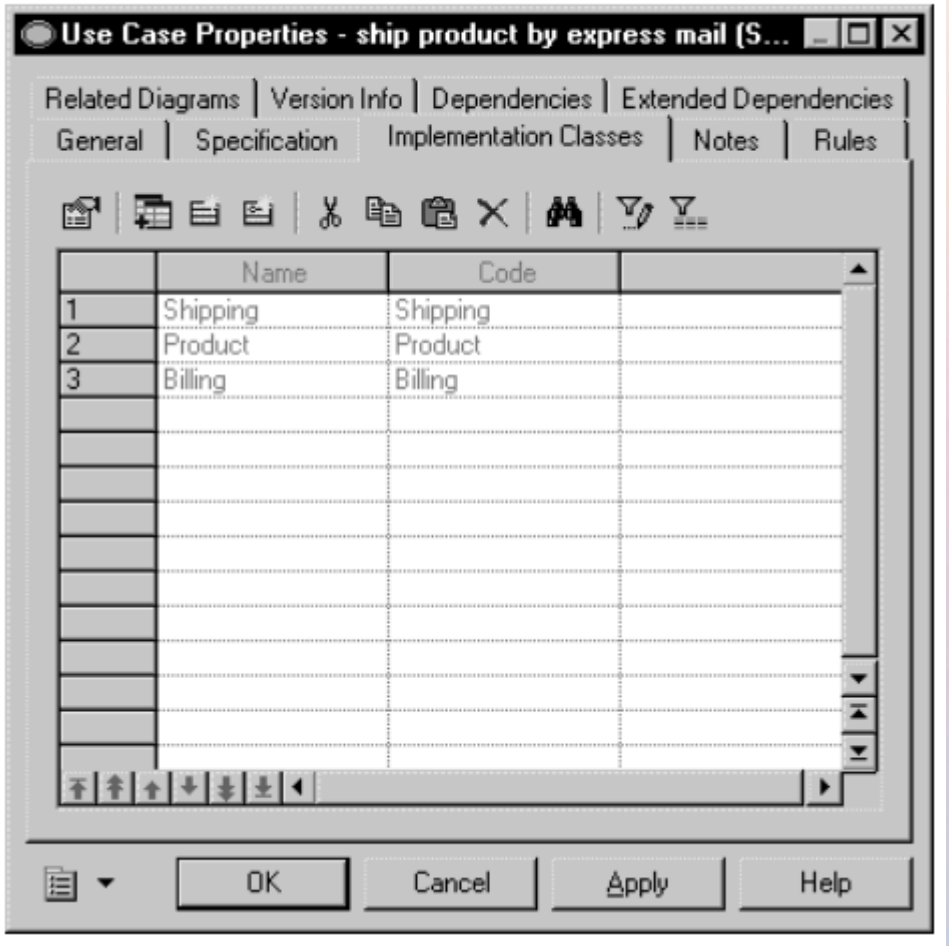
For example, a use case **ship product by express mail** needs the classes **Shipping**, **Product**, and **Billing** to perform its task. They are shown as related to the use case in the drawing below.



5.2A.2. ĐỊNH NGHĨA CÁC USE CASE TRONG BIỂU ĐỒ CA

Ví dụ (tt)

The implementation classes are the following:



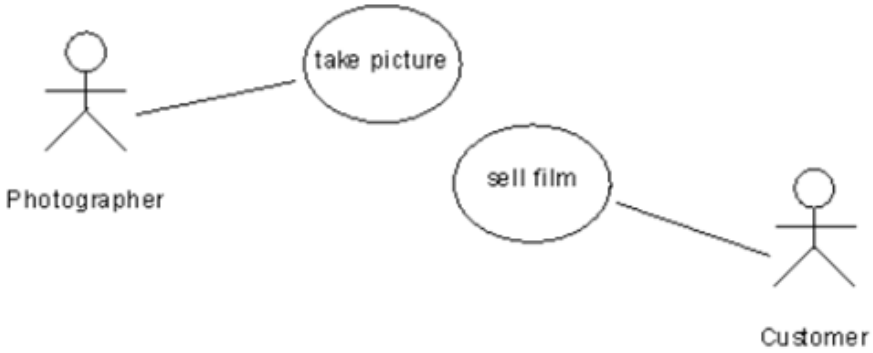
5.2A.3. ĐỊNH NGHĨA CÁC ACTOR (TÁC NHÂN) TRONG B.Đ CA

Defining actors in a use case diagram

The system you are describing interacts with **actors**. An actor characterizes an outside user, or related set of users that interact with a system.

The most obvious candidates for actors are the humans in the system, but it can also be a system by itself.

Example



an actor can be a **primary actor**

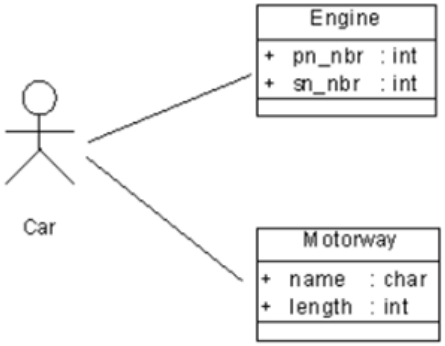
An actor can be a **secondary actor**

5.2A.3. ĐỊNH NGHĨA CÁC ACTOR (TÁC NHÂN) TRONG B.Đ CA

Actor implementation classes

The Implementation Classes page shows the list of classes and interfaces used to implement an actor. An actor can be a human being (person, partner) or a machine, or process (automated system). When analyzing what an actor must do, it is necessary to identify the classes and interfaces that need to be created for the actor to perform his task.

For example, an actor Car needs the classes Engine and Motorway to perform its task. They are shown as related to the actor in the following figure:



5.2A.3. ĐỊNH NGHĨA CÁC ACTOR (TÁC NHÂN) TRONG B.Đ CA

5.2A.4. ĐỊNH NGHĨA CÁC ASSOCIATION TRONG B.ĐỒ CA

Defining associations in a use case diagram

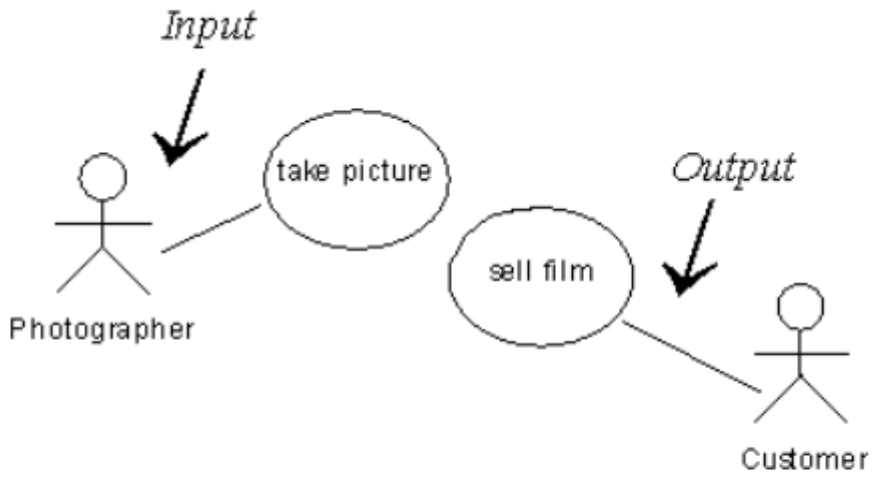
An **association** is a unidirectional relationship that describes a link between objects.

You can create an association between:

- An actor and a use case
- A use case and an actor

Associations in a use case diagram are very close to those in the class diagram. However, it is not possible to specify roles, cardinalities and navigability for associations.

Example



An association between an actor and a use case is an **input** association. An association between a use case and an actor is an **output** association.

5.2A.5. ĐỊNH NGHĨA GENERALIZATION TRONG B.ĐỒ CA

Defining generalizations in a use case diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

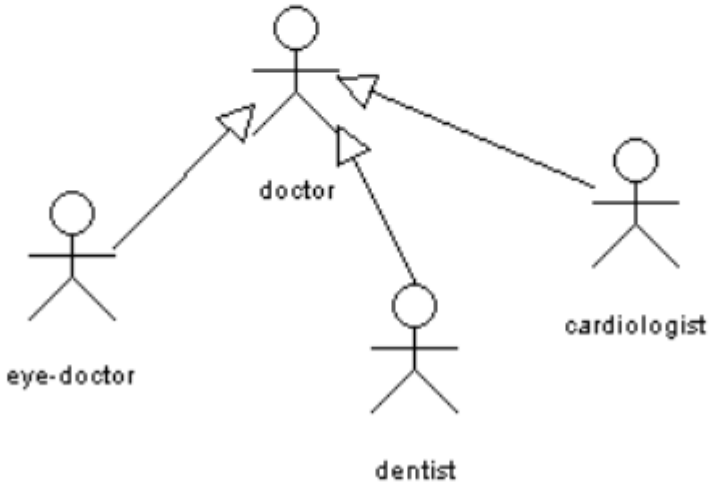
A generalization can be created between:

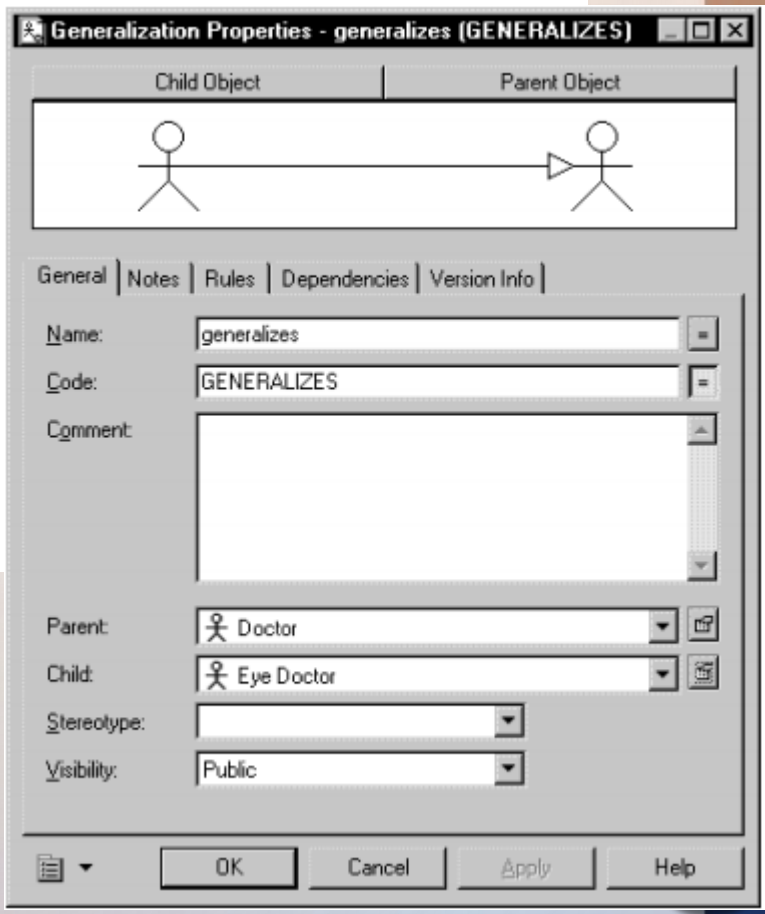
- Two actors
- Two use cases

5.2A.5. ĐỊNH NGHĨA GENERALIZATION TRONG B.ĐỒ CA

Example

For example two or more actors may have similarities, and communicate with the same set of use cases in the same way. This similarity is expressed with generalization to another actor. In this case, the child actors inherit the roles, and relationships to use cases held by the parent actor. A child actor includes the attributes and operations of its parent.





5.2A.6. ĐỊNH NGHĨA DEPENDENCY TRONG B.ĐỒ CA

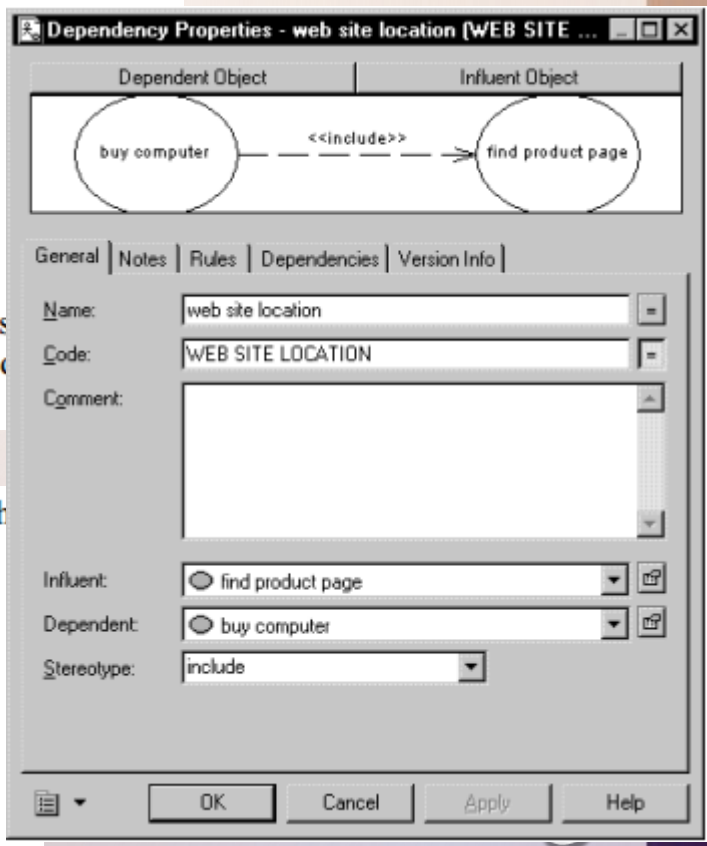
Defining dependencies in a use case diagram

A dependency is a semantic relationship between two modeling elements, in which a change to one modeling element (the influent element) may affect the semantics of the other modeling element (the dependent element).

A dependency can be created between:

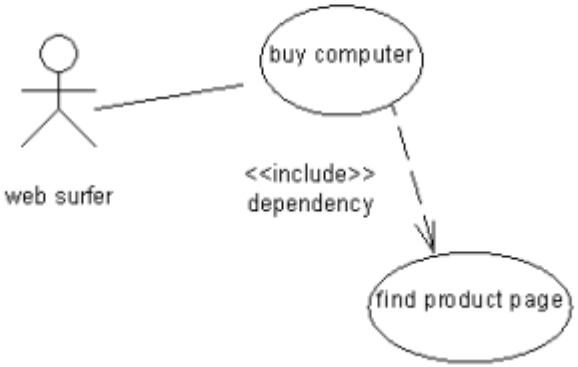
- An actor and a use case (and vice versa)
- Two actors
- Two use cases

A dependency created between two use cases, or two actors, means
former object "uses" the latter, in other words, the behavior define
object employs the behavior defined for the other.



Example

Buying a computer from a web site involves the activity of finding th
product page within the seller's web site:



5.2B. XÂY DỰNG BIỂU ĐỒ LỚP (CLASS)

5.2B.1. Cơ bản về biểu đồ LỚP

5.2B.2. Định nghĩa các LỚP trong biểu đồ lớp

5.2B.3. Định nghĩa các GIAO DIỆN trong biểu đồ lớp

5.2B.4. Định nghĩa các THUỘC TÍNH trong biểu đồ lớp

5.2B.5. Định nghĩa các ĐỊNH DANH trong biểu đồ lớp

5.2B.6. Định nghĩa các OPERATION trong biểu đồ lớp

5.2B.7. Định nghĩa các LIÊN KẾT trong biểu đồ lớp

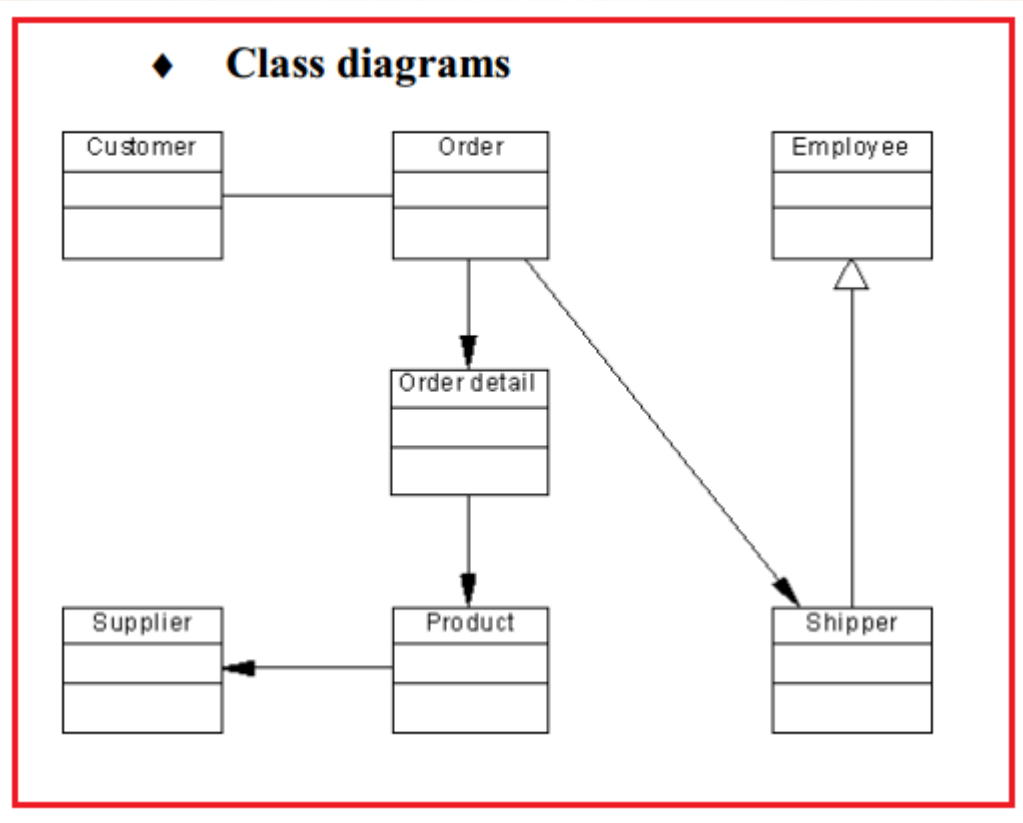
5.2B.8. Định nghĩa Generalization trong biểu đồ lớp

5.2B.9. Định nghĩa Dependency trong biểu đồ lớp

5.2B.10. Định nghĩa Realization trong biểu đồ lớp

5.2B.11. Định nghĩa Domain và Check Parameter trong b.đồ lớp

5.2B. XÂY DỰNG BIỂU ĐỒ LỚP (CLASS)



5.2B.1. CƠ BẢN VỀ BIỂU ĐỒ LỚP

Class diagram basics

A **class diagram** defines the static structure of the model, it contains a set of packages, classes, interfaces, and their relationships. These objects together form a class structure that is the logical design view of all, or part of a software system.

Defining a class diagram

The class diagram is one of the diagrams of the Unified Modeling Language (UML). It is a static design view of a system.

The class diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of the objects defined in the system. The objects are the classes, packages, and the relationships among them.

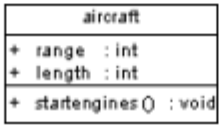
5.2B.2. ĐỊNH NGHĨA CÁC LỚP TRONG BIỂU ĐỒ LỚP

Defining classes in a class diagram

A **class** is a description of a set of objects that have a similar structure and behavior, and share the same attributes, operations, relationships, and semantics. Structure is described by attributes and associations. Behavior is described by operations.

Classes are the main building blocks of an OOM. Classes, and the relationships that you create between them, form the basic structure of an OOM. A class defines a concept within the application being modeled, such as a physical thing (like a car), a business thing (like an order) a logical thing (like a broadcasting schedule), an application thing (like an OK button), or a behavioral thing (like a task).

The following example shows the class Aircraft with its attributes (range and length) and operation (startengines).



5.2B.2. ĐỊNH NGHĨA CÁC LỚP TRONG BIỂU ĐỒ LỚP

From the property sheet of a class you access the following information about the class:

- Attributes. These are named properties of the class that describe its characteristics. See section Defining attributes in a class diagram.
- Identifiers. These are class attributes, or combinations of class attributes, whose values uniquely identify each occurrence of the class. See section Defining identifiers in a class diagram.
- Operations. These are specification of a query that an interface may be called to execute. See section Defining operations in a class diagram.
- Associations. These display the list of migrated attributes proceeding from navigable associations connected to the class. See section Viewing the migrated attributes of a class.
- Inner classifiers. These display the list of inner classes and interfaces of the current class. See section Defining inner classifiers.
- Script. This allows you to customize your scripts. See section Customizing scripts for classes and interfaces.
- Preview. This allows you to preview the code of the class. See section Previewing the code of a class or interface.

5.2B.2. ĐỊNH NGHĨA CÁC LỚP TRONG BIỂU ĐỒ LỚP

Class type

A type is used to define the class implementation. For example, you can use the type to implement a task (middle tier, business object), or a storage type.

You can declare a class to be one of the following types:

- Business Object
- Class
- Storage
- Utility
- Visual Object
- JavaBean

Class visibility

The visibility of a class is the way it can be seen by other objects. When a class is visible to another object, it may influence the structure or behavior of the object, or similarly, the other object can affect the properties of the class.

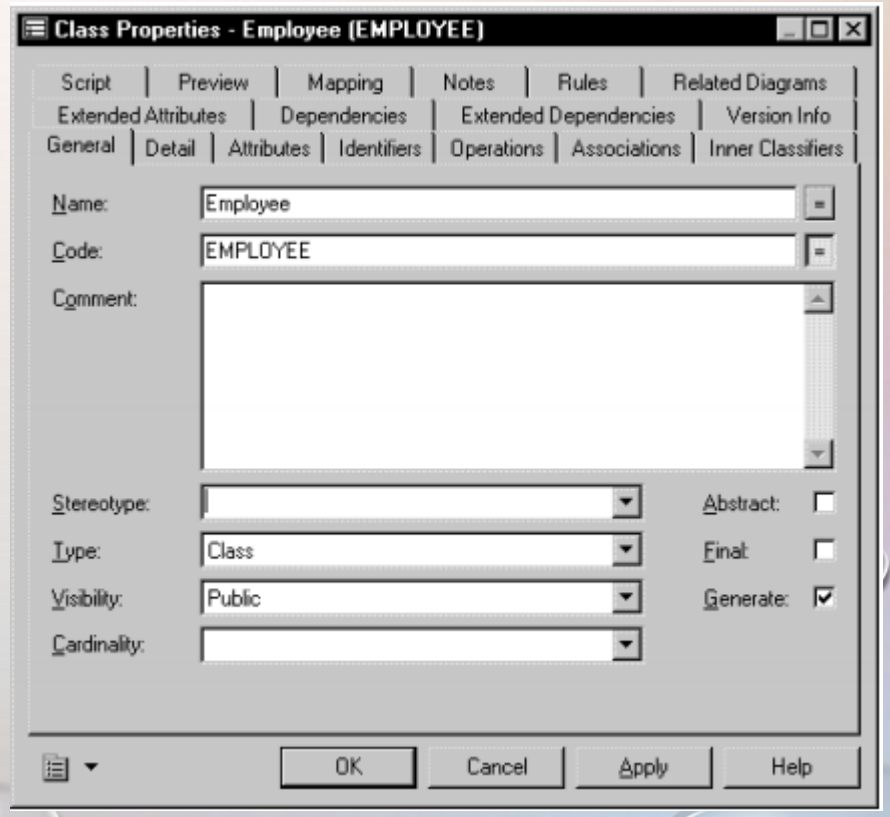
Property	Visible
Private	Only to the class itself
Protected	Only to the class and its derived objects
Package	To all objects contained within the same package
Public	To all objects in the model

5.2B.2. ĐỊNH NGHĨA CÁC LỚP TRONG BIỂU ĐỒ LỚP

Class cardinality

The cardinality of a class specifies the number of instances a class can have.

Cardinality	Number of instances
0..1	None to one
0..*	None to an unlimited number
1..1	One to one
1..*	One to an unlimited number
*	Unlimited number



5.2B.2. ĐỊNH NGHĨA CÁC LỚP TRONG BIỂU ĐỒ LỚP

Creating a class from an interface

You can create a class in a diagram from the contextual menu of an interface. This allows you to inherit all the operations of the interface, including the getter and setter operations.

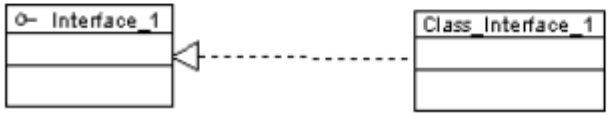
❖ **To create a class from an interface:**

1 Right-click an interface in the diagram to display the interface contextual menu.

2 Select Create Class from the contextual menu.

A new class and a new realization are created in the diagram. They are also visible from the Browser.

The class is suffixed by the name of the interface.



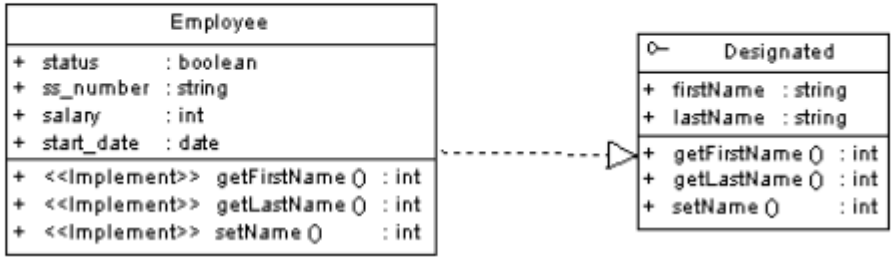
5.2B.3. ĐỊNH NGHĨA CÁC GIAO DIỆN TRONG BIỂU ĐỒ LỚP

Defining interfaces in a class diagram

An interface is similar to a class but it is used to define the specification of a behavior. It is a collection of operations specifying the externally visible behavior of a class. It has no implementation of its own.

An interface includes the signatures of the operations. It specifies only a limited part of the behavior of a class. A class can implement one or more interfaces.

A class must implement all the operations in this interface to realize the interface. The following example shows an interface (Designated) realized by a class (Employee).

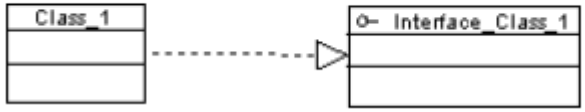


5.2B.3. ĐỊNH NGHĨA CÁC GIAO DIỆN TRONG BIỂU ĐỒ LỚP

Creating an interface from a class

You can create an interface in a diagram from the contextual menu of a class. This allows you to inherit all the operations of the class, including the getter and setter operations.

The interface is suffixed after the name of the class.



5.2B.4. ĐỊNH NGHĨA CÁC THUỘC TÍNH TRONG BIỂU ĐỒ LỚP (1)

Defining attributes in a class diagram

An attribute is a named property of a class (or an interface) describing its characteristics.

A class or an interface may have none or several attributes. Each object in a class has the same attributes, but the values of the attributes may be different.

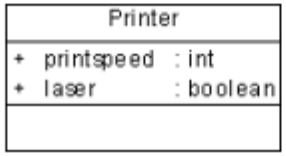
Attribute names within a class must be unique. You can give identical names to two or more attributes only if they exist in different classes.

You can define attributes for the following elements:

- Class
- Interface
- Identifier

Example

The class Printer, contains two attributes: printspeed and laser:



5.2B.4. ĐỊNH NGHĨA CÁC THUỘC TÍNH TRONG BIỂU ĐỒ LỚP (2)

Attribute changeability in a class diagram

The changeability of an attribute has the following values:

Changeability	Value
Changeable	The value can be changed
Read-only	Prevents the creation of a setter operation (a setter is created in the method inside the class)
Frozen	Constant
Add-only	Allows you to add a new value only

5.2B.5. ĐỊNH NGHĨA CÁC ĐỊNH DANH TRONG BIỂU ĐỒ LỚP

Defining identifiers in a class diagram

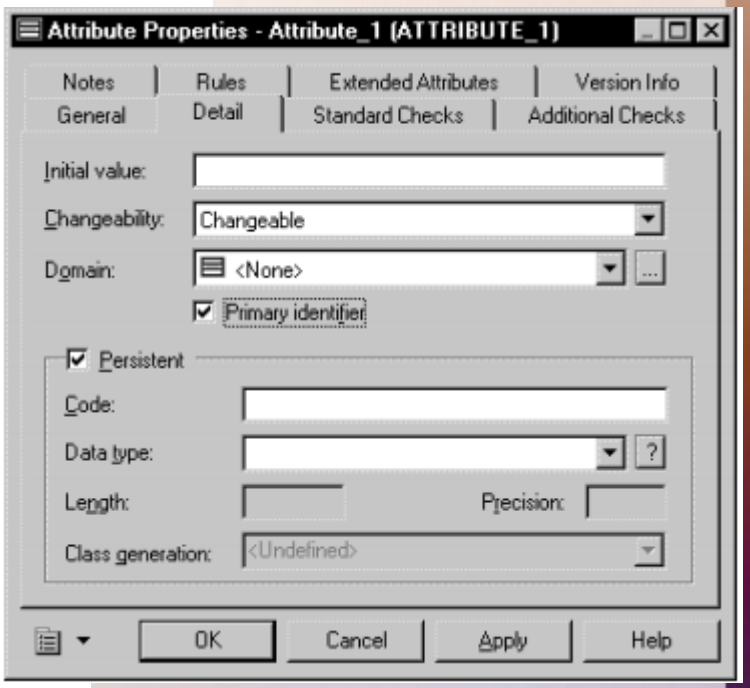
An identifier is a class attribute, or a combination of class attributes, whose values uniquely identify each occurrence of the class. It is used during intermodel generation when you generate a CDM or a PDM into an OOM, the CDM identifier and the PDM primary or alternate keys become identifiers in the OOM.

Each class can have at least one identifier. Among identifiers, t
identifier is the main identifier of the class. This identifier corre
primary key in the PDM.

When you create an identifier, you can attach attributes or busi
You can also define one or several attributes as being primary i
the class.

Identifier example

The social security number for a class employee is the primary
this class.



Creating a primary identifier

There are several ways to create a primary identifier:

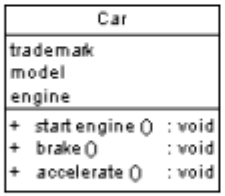
- Create a primary identifier while you create the class attributes
- Define the primary identifier from the list of identifiers

5.2B.6. ĐỊNH NGHĨA CÁC OPERATION TRONG BIỂU ĐỒ LỚP (1)

Defining operations in a class diagram

An operation is a named specification of a service that can be requested from any of a class's objects to affect behavior. It is a specification of a query that an object may be called to execute. A class may have any number of operations or no operations at all.

In the following example, the class Car, has 3 operations: start engine, brake, and accelerate.



Operations have a name and a list of parameters. Several operations can have the same name within the same class if their parameters are different.

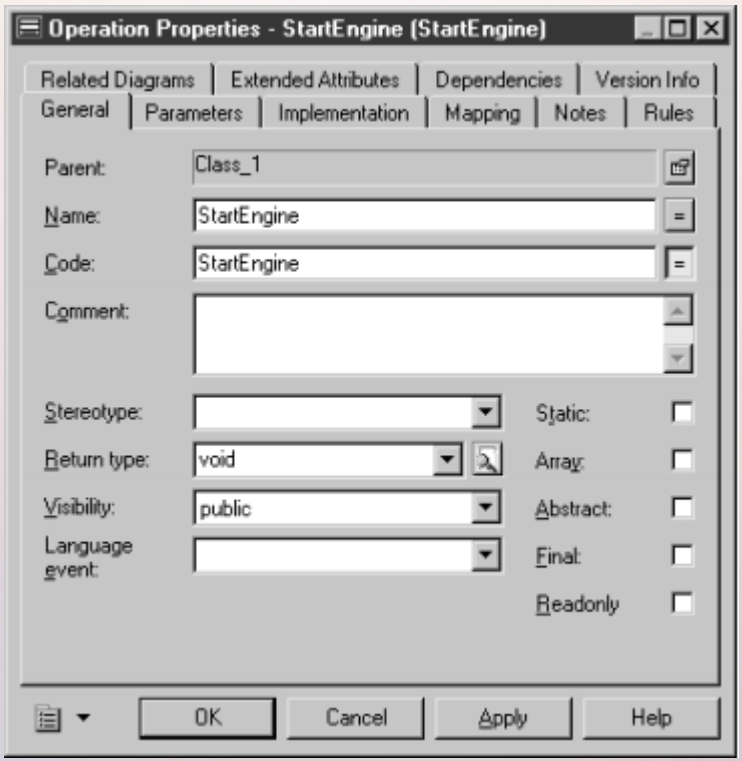
If you want to use operations that are specific to EJB components, you should refer to chapter Working with platform dependent components in this manual.

5.2B.6. ĐỊNH NGHĨA CÁC OPERATION TRONG BIỂU ĐỒ LỚP (2)

Creating an operation

You create an operation from the class or interface property sheet. To display the operation property sheet, you can:

- Double-click the class or interface symbol
- Right-click the class or interface node in the Browser and select Properties



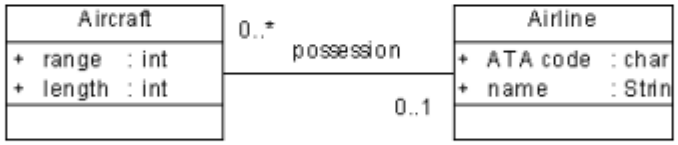
5.2B.7. ĐỊNH NGHĨA CÁC LIÊN KẾT TRONG BIỂU ĐỒ LỚP (1)

Defining associations in a class diagram

An association represents a structural relationship between classes or between a class and an interface. It is drawn as a solid line between pairs of classes, or classes and interfaces.

You can define an association between two classes, or between a class and an interface.

Example



An association must have a name but the name you give to the association is less important if you use association roles.

Association Roles

Each side of an association may have a name called a role, the role describes the function of a class as viewed by the opposite class.

5.2B.7. ĐỊNH NGHĨA CÁC LIÊN KẾT TRONG BIỂU ĐỒ LỚP (2)

Association role in a class diagram

Each end of an association is called a **role**. You can define its multiplicity and define one of the two roles as an aggregation or a composition. You can also define a visibility for the association.

Property	Description
Multiplicity	Minimum and maximum number of instances of the association
Ordering	The association is included in the ordering which sorts the list of associations by their order of creation.
Visibility	Visibility of the role of the association whose value denotes how it is seen outside its enclosing namespace
Navigable	Indicates whether or not information can be transmitted between the two objects linked by the relationship
Aggregation	A form of association that specifies a part-whole relationship between a class and an aggregate class (example: a car has an engine and wheels)
Composition	A form of aggregation but with strong ownership and coincident lifetime of parts by the whole; the parts live and die with the whole (example: an invoice and its invoice line)

The aggregation symbol in a diagram is the following:

The composition symbol in a diagram is the following:

5.2B.7. ĐỊNH NGHĨA CÁC LIÊN KẾT TRONG BIỂU ĐỒ LỚP (3)

Association multiplicity in a class diagram

The allowable cardinalities of a role are called multiplicity. Multiplicity indicates the maximum and minimum cardinality that a role can have.

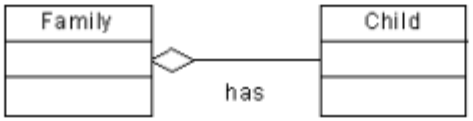
Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

5.2B.7. ĐỊNH NGHĨA CÁC LIÊN KẾT TRONG BIỂU ĐỒ LỚP (4)

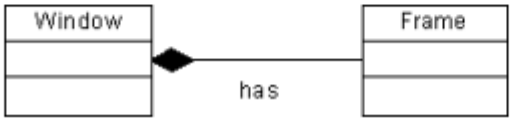
Aggregation/composition

You can define one of the roles of an association as being either an aggregation or a composition in the Aggregation/Composition groupbox.

An **aggregation** is a type of association in which one class represents a larger thing (a whole) made of smaller things (the parts). This special association is a "has-a" link. It means that an object of the whole has objects of the part. In the following example, the family is the whole that can contain children.



A **composition** is also a whole/part type of association in which the parts are strongly tied to the whole. In a composition, an object may be a part of only one composite at a time, and the composite object manages the creation and destruction of its parts. In the following example, the frame is a part of a window. If you destroy the window object, the frame part also disappears.



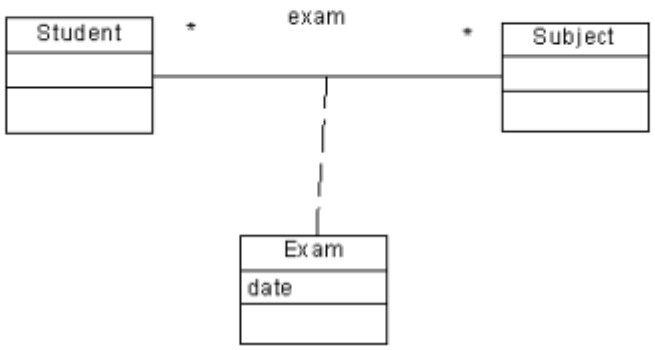
5.2B.7. ĐỊNH NGHĨA CÁC LIÊN KẾT TRONG BIỂU ĐỒ LỚP (5)

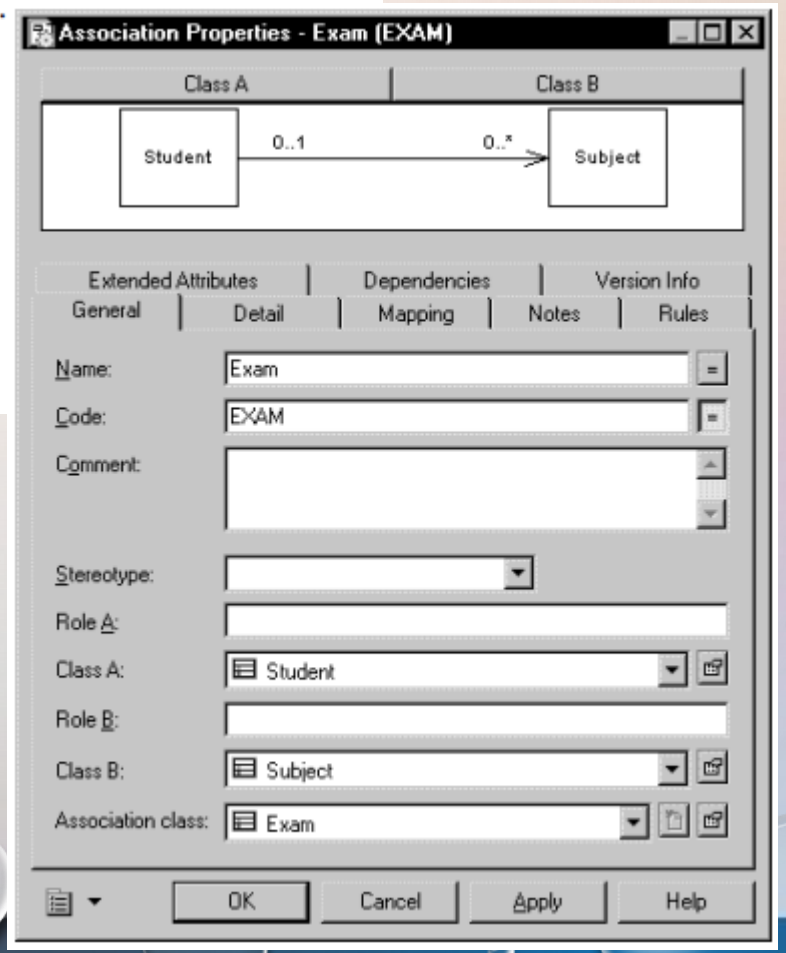
Creating an association class

You can add properties to an association between classes or interfaces by creating an **association class**. It is used to further define the properties of an association by adding attributes and operations to the association.

Example

In the following example, the classes Student and Subject are related by an association exam. However, this association does not specify the date of the exam. You can create an association class called Exam that will indicate additional information concerning the association.





5.2B.8. ĐỊNH NGHĨA GENERALIZATION TRONG BIỂU ĐỒ LỚP (6)

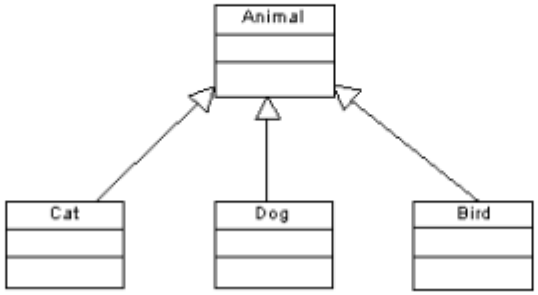
Defining generalizations in a class diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

For example, an animal is a more general concept than a cat, a dog or a bird. Inversely, a cat is a more specific concept than an animal.

Example

Animal is a super class. Cat, Dog and Bird are sub-classes of the super class.



5.2B.8. ĐỊNH NGHĨA GENERALIZATION TRONG BIỂU ĐỒ LỚP (1)

Generalizations exist in the following diagrams:

- Class diagram

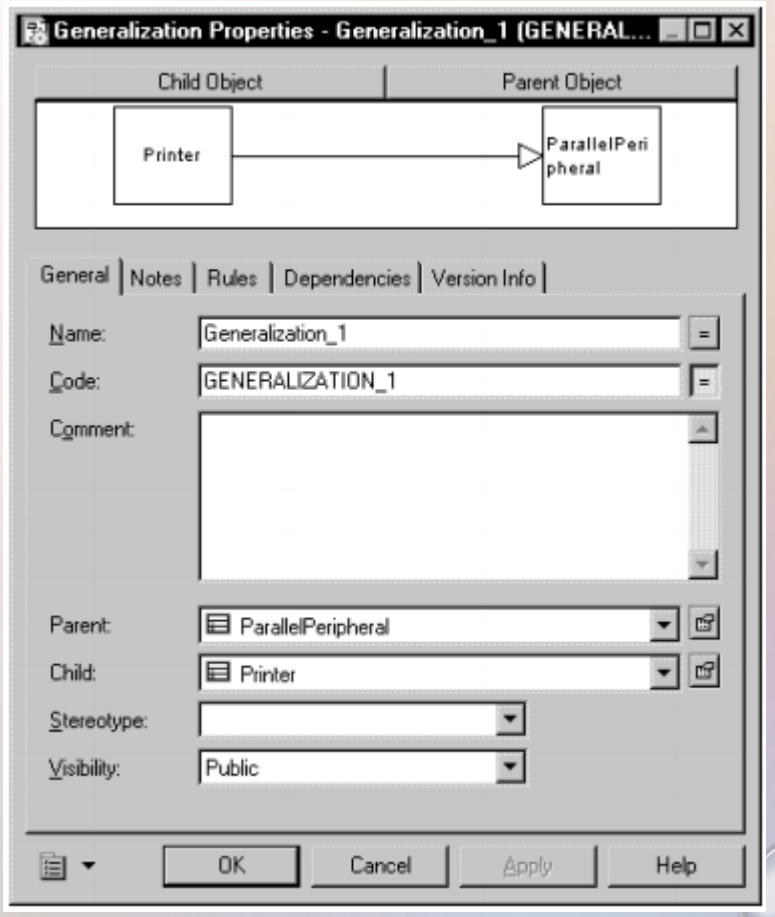
  A generalization can be created between:

  Two classes

  Two interfaces

  You can also create a generalization between a shortcut of a class and a class, a shortcut of an interface and an interface. Whenever a link is oriented, only the parent object can be the shortcut.

- Use case diagram
- Component diagram

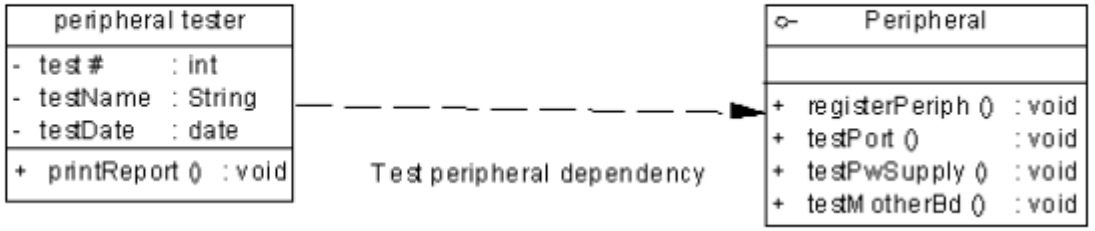


5.2B.9. ĐỊNH NGHĨA DEPENDENCY TRONG BIỂU ĐỒ LỚP

Defining dependencies in a class diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

The dependency relationship indicates that one object in a diagram uses the services or facilities of another object. You can also define dependencies between a package and a modeling element.



Dependencies exist in the following diagrams:

- Class diagram

  A dependency can be created between:

  - A class and an interface (and vice versa)
  - Two classes
  - Two interfaces
- Use case diagram
- Component diagram
- Deployment diagram
- Object diagram

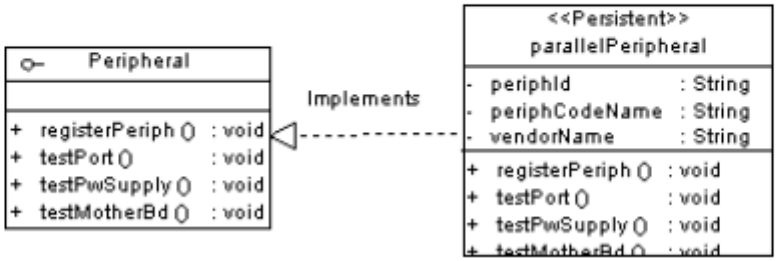
5.2B.10. ĐỊNH NGHĨA REALIZATION TRONG BIỂU ĐỒ LỚP

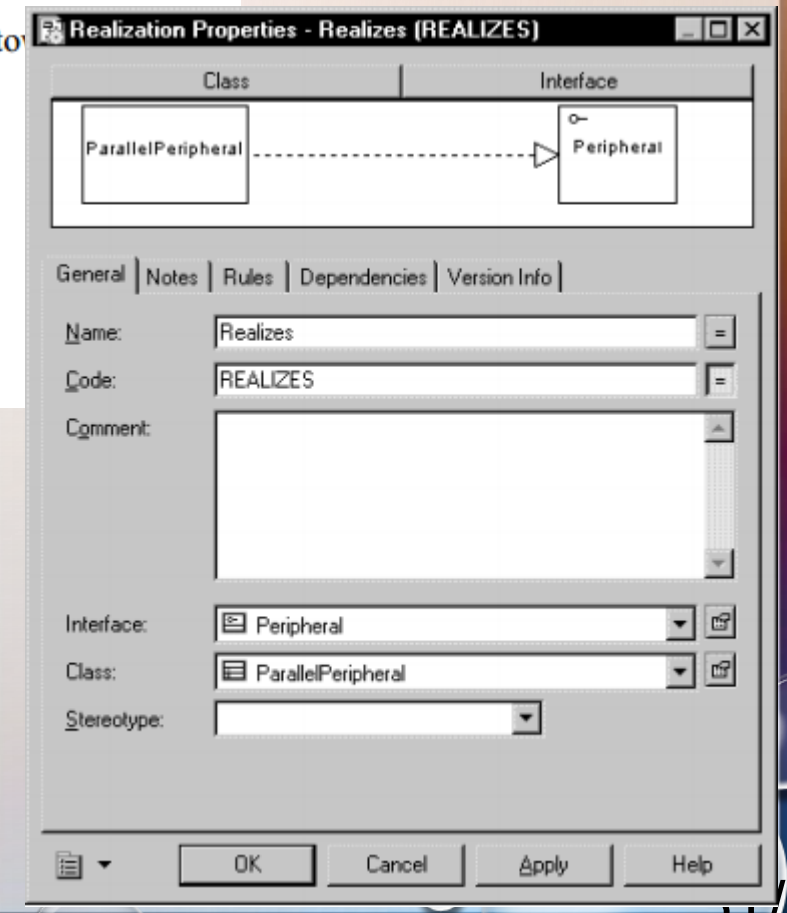
Defining realizations in a class diagram

A realization is a relationship between a class and an interface. In a realization, the class implements the methods specified in the interface. The interface is called the specification element and the class is called the implementation element.

You can also create a realization between a shortcut of an interface and a class. Whenever the link is oriented, only the parent object can be the shortcut.

The arrowhead at one end of the realization always points to
interface.





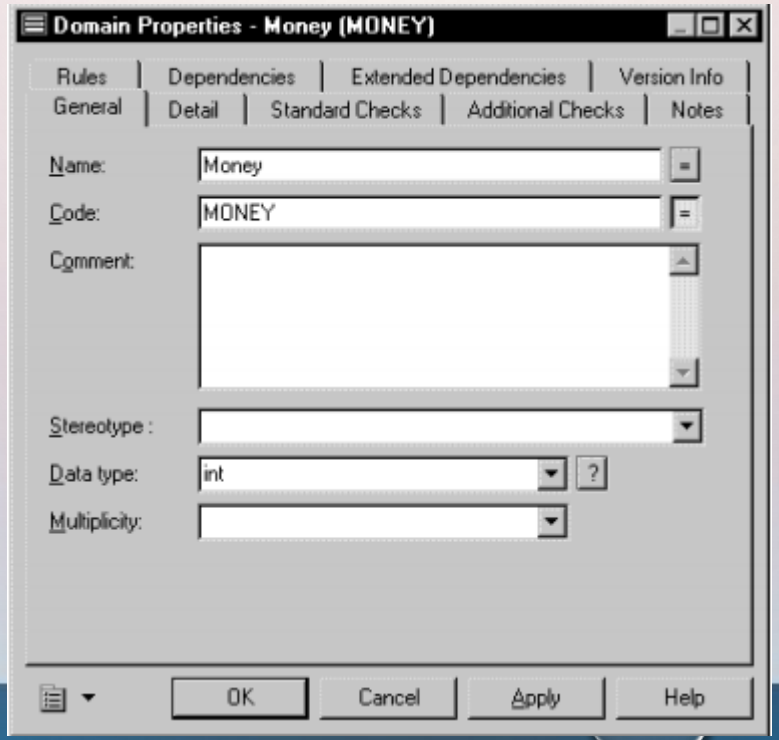
5.2B.11. ĐỊNH NGHĨA DOMAIN VÀ CHECK PARAMETER TRONG B.ĐỒ LỚP

Defining domains in an OOM

Domains define the set of values for which an attribute is valid. They are used to enforce consistent handling of data across the system. Applying domains to attributes makes it easier to standardize data characteristics for attributes in different classes.

In an OOM, you can associate the following information with a domain:

- Data type
- Check parameters
- Business rules



5.2B.11. ĐỊNH NGHĨA DOMAIN VÀ CHECK PARAMETER TRONG B.ĐỒ LỚP

Updating attributes using a domain in an OOM

When you modify a domain, you can choose to automatically update the following properties for attributes using the domain:

- Data type
- Check parameters
- Business rules

Defining check parameters in an OOM

Check parameters are sets of conditions which data must satisfy to remain valid. They are attached to attributes and domains. They are principally used in a CDM or a PDM. They are of importance only when importing a CDM or importing a PDM into an OOM. They are not used in the OOM alone.

There are three types of check parameters you can therefore use in an OOM:

Parameter type	Description
Standard parameters	Common data constraints defining a data range
Additional check parameters	SQL expression defining a data constraint using variables instantiated with standard parameter values
Validation rule	Business rule defined as a server expression

5.2C. XÂY DỰNG BIỂU ĐỒ ĐỐI TƯỢNG (OBJECT)

5.2C.1. Cơ bản về biểu đồ ĐỐI TƯỢNG (ĐT)

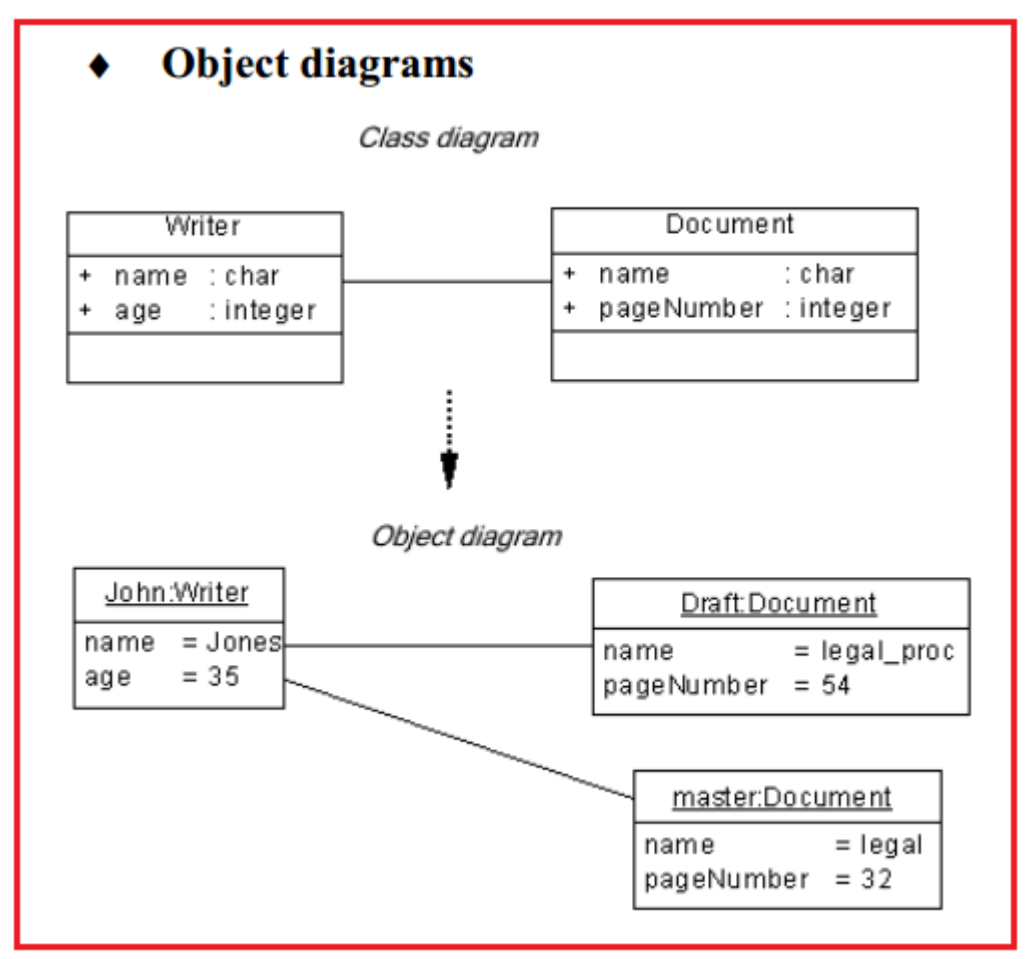
5.2C.2. Định nghĩa các ĐỐI TƯỢNG trong biểu đồ ĐT

5.2C.3. Định nghĩa các THUỘC TÍNH trong biểu đồ ĐT

5.2C.4. Định nghĩa các INSTANCE liên kết trong biểu ĐT

5.2C.5. Định nghĩa Dependency trong biểu đồ ĐT

5.2C. XÂY DỰNG BIỂU ĐỒ ĐỐI TƯỢNG (OBJECT)



5.2C.1. CƠ BẢN VỀ BIỂU ĐỒ ĐỐI TƯỢNG (ĐT) (1)

Object diagram basics

The **object diagram** and the class diagram constitute the two UML static structure diagrams. The object diagram is considered a diagram of instances of classes.

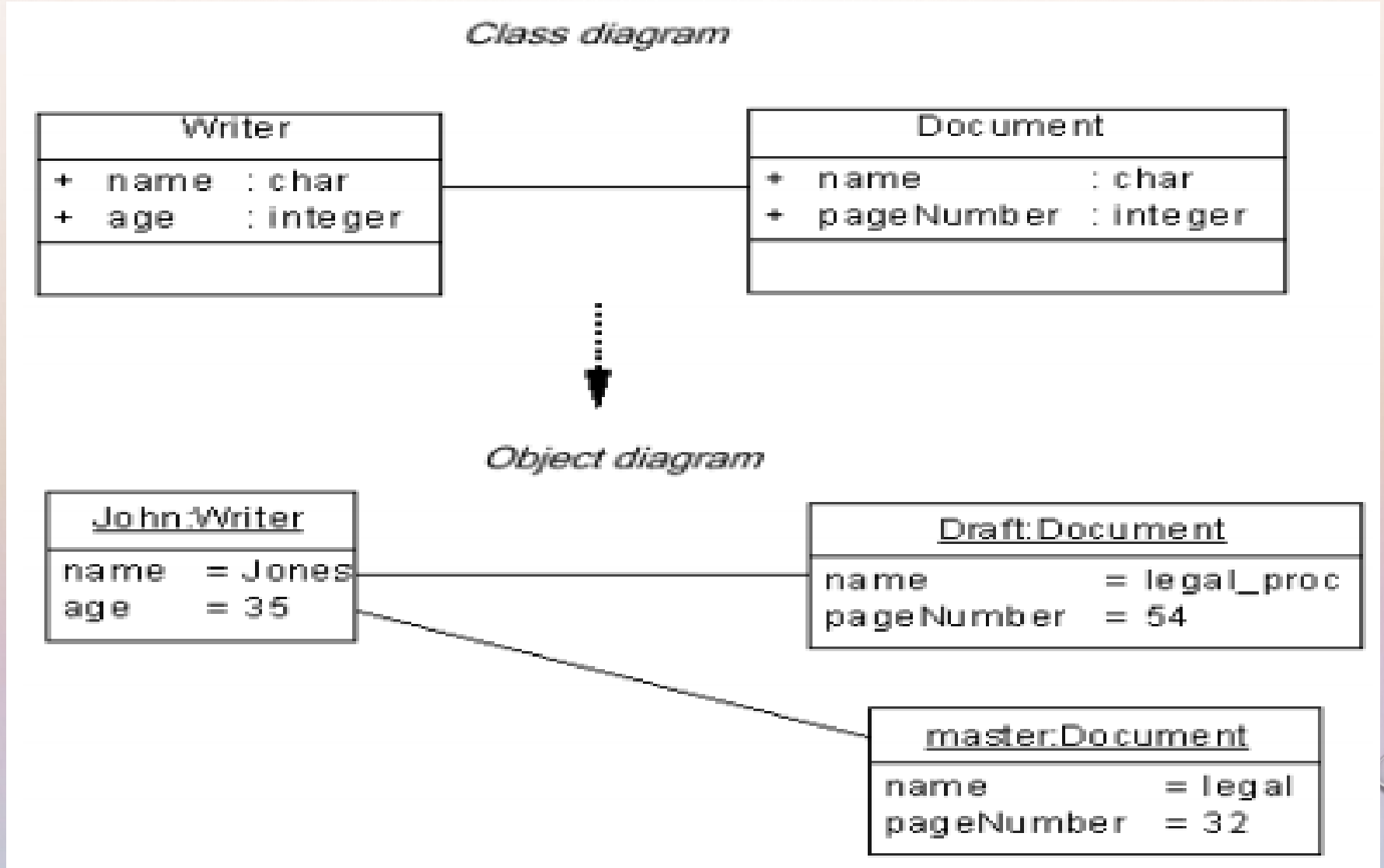
Defining an object diagram

The object diagram is one of the diagrams of the Unified Modeling Language (UML). The object diagram describes the structure of model elements and not their behavior or temporary information. It is a diagram of instances: it represents instances of class (objects), instances of association (instance links), and dependencies. However, it does not represent inheritance relationships.

Why build an object diagram?

The object diagram can be used for analysis purpose: constraints between classes that are not classically represented in a class diagram can typically be represented in an object diagram.

5.2C.1. CƠ BẢN VỀ BIỂU ĐỒ ĐỐI TƯỢNG (ĐT) (2)



5.2C.2. ĐỊNH NGHĨA CÁC ĐỐI TƯỢNG TRONG BIỂU ĐỒ ĐT (1)

Defining objects in an object diagram

At the conceptual level, an **object** is an element defined as being part of the system described. It represents an object that has not yet been instantiated because the classes are not yet clearly defined at this stage.

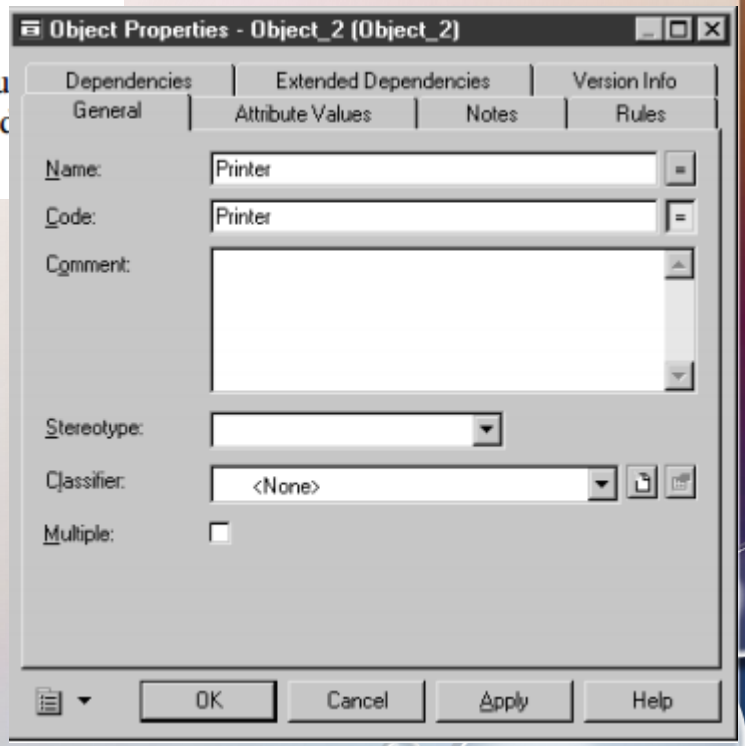
Three possible situations can be represented:

When an object is not an instance of a class it has only a name.

When an object is an instance of a class it has a name and a class.

When an object is an instance of a class but is actually representing any or all instances of a class it has a class but no name.

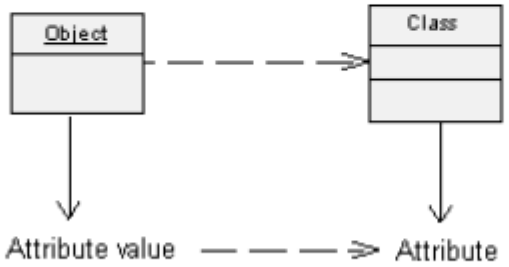
In the object diagram, an object instance of a class can display the valu
attributes defined on the class. When the class is deleted, the associated
objects are not deleted.



5.2C.3. ĐỊNH NGHĨA CÁC THUỘC TÍNH TRONG BIỂU ĐỒ ĐT (2)

Defining attribute values in an object diagram

An attribute value represents an instance of a class attribute. It refers to an attribute of the class from which the current object is an instance, or it can refer to an attribute inherited from a parent of the class. An attribute value is stored as text.



An attribute value is created when you assign a value to an attribute. There is no symbol of attribute value, there is no property sheet either.

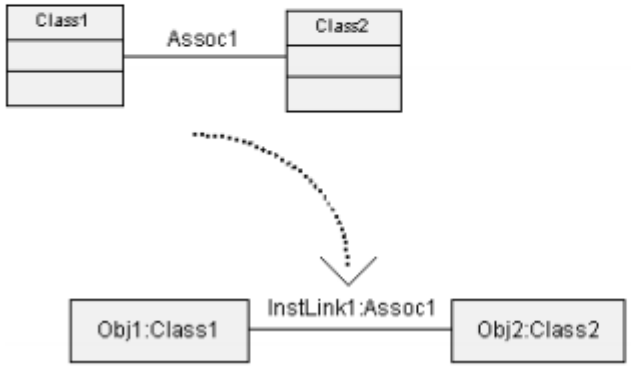
5.2C.4. CÁC INSTANCE LIÊN KẾT TRONG BIỂU ĐT (1)

Defining instance links in an object diagram

An **instance link** represents a connection between two objects. It is drawn as a solid line between two objects.

The instance link in the object diagram shares the same concept as in the collaboration diagram.

Instance links have a strong relationship with associations of the class diagram: associations between classes, or associations between a class and an interface can become instance links (instances of associations) between objects in the object diagram. Moreover, the instance link symbol in the object diagram is similar to the association symbol in the class diagram, except that the instance link symbol has no cardinalities.

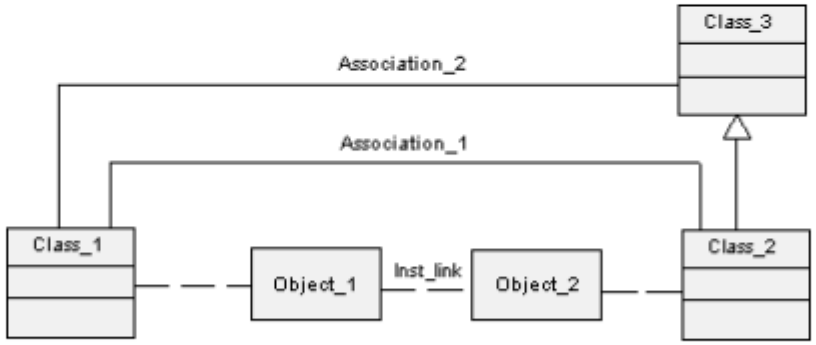


5.2C.4. CÁC INSTANCE LIÊN KẾT TRONG BIỂU ĐT (2)

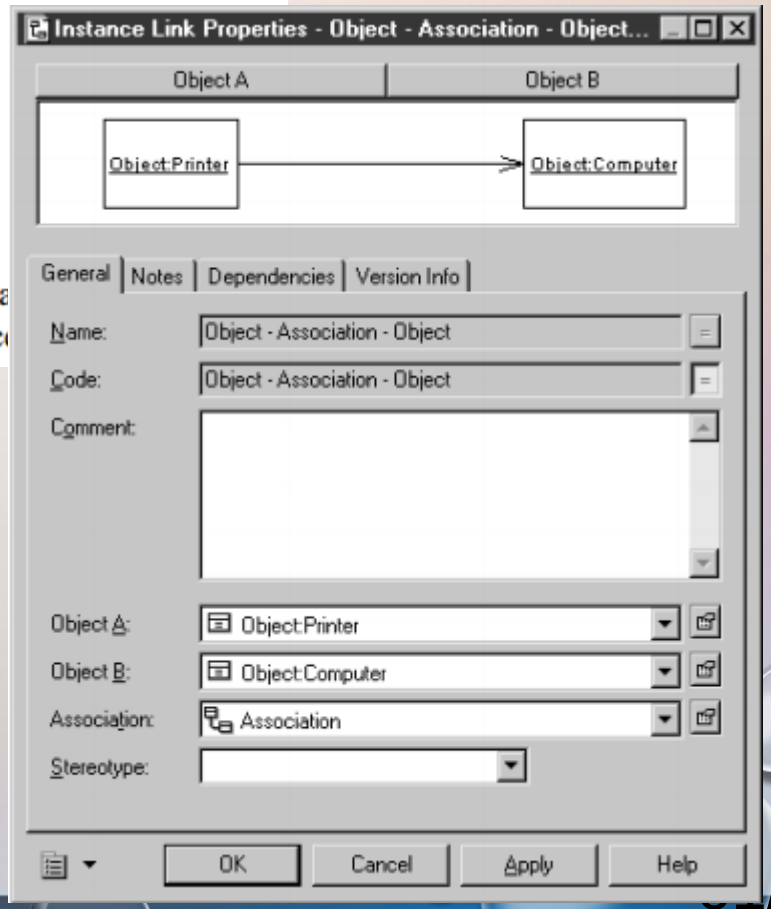
Example

The following figure shows Object_1 as instance of Class_1, and Object_2 as instance of Class_2. They are linked by an instance link. It shows Class_1 and Class_2 linked by an association. Moreover, since Class_2 is associated with Class_1 and also inherits from Class_3, there is an association between Class_1 and Class_3.

The instance link between Object_1 and Object_2 in the figure can represent Association_1 or Association_2.



You can also use shortcuts of associations, however you ca
model to which the shortcut refers is open in the workspac

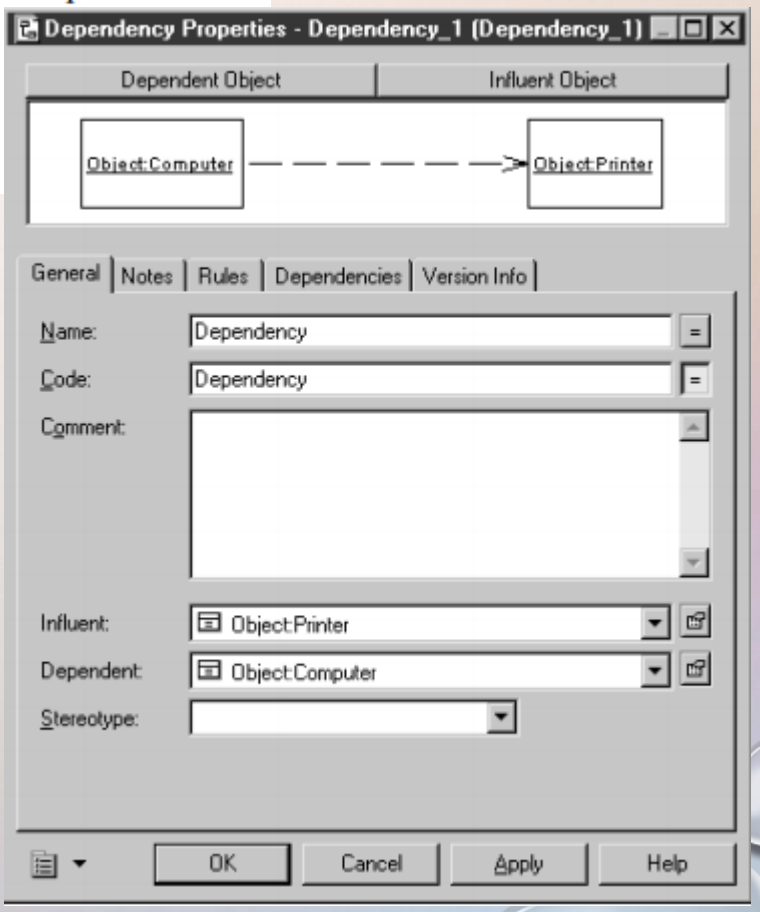


5.2C.5. ĐỊNH NGHĨA DEPENDENCY TRONG BIỂU ĐỒ ĐT

Defining dependencies in an object diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

The dependency relationship indicates that one object in a diagram uses the services or facilities of another object. You can also define dependencies between a package and a modeling element.



5.3. XÂY DỰNG BIỂU ĐỒ HỢP TÁC, TUẦN TỰ

5.3A. Xây dựng biểu đồ HỢP TÁC

5.3B. Xây dựng biểu đồ TUẦN TỰ

5.3A. XÂY DỰNG BIỂU ĐỒ HỢP TÁC

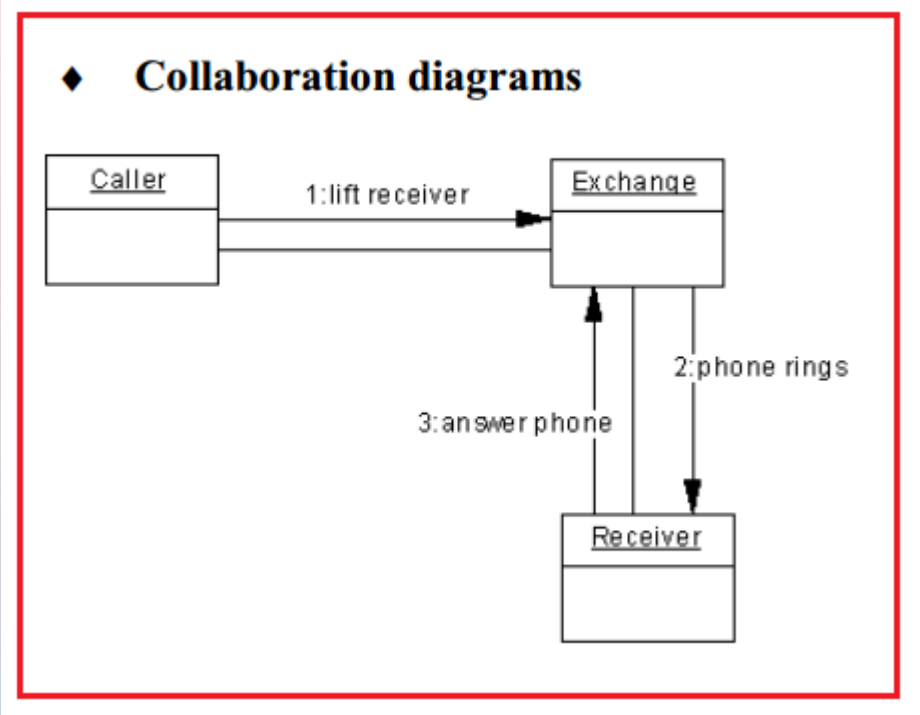
5.3A.1. Cơ bản về biểu đồ HỢP TÁC (HTAC)

5.3A.2. Định nghĩa các TÁC NHÂN trong biểu đồ HTAC

5.3A.3. Định nghĩa các ĐỐI TƯỢNG trong biểu đồ HTAC

5.3A.4. Định nghĩa các INSTANCE liên kết trong biểu HTAC

5.3A.5. Định nghĩa các MESSAGE trong biểu đồ HTAC



5.3A.1. CƠ BẢN VỀ BIỂU ĐỒ HỢP TÁC (HTAC) (1)

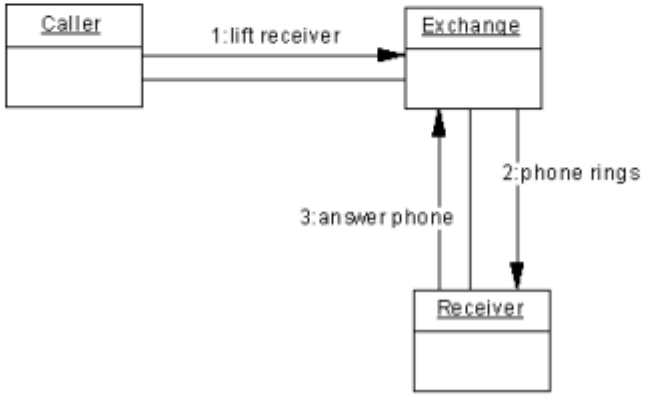
Collaboration diagram basics

The **collaboration diagram** is part of the Unified Modeling Language (UML). It is one of the two interaction diagrams, the other being the sequence diagram.

Defining a collaboration diagram

The collaboration diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows actors, objects (instances of classes) and their communication links (called instance links), as well as messages sent between them.

A collaboration diagram shows these elements performing a particular goal (a functionality of the system). It may be used to illustrate an execution of an operation, a use-case execution, or simply an interaction scenario in the system. It designs an example of interactions between objects.



5.3A.1. CƠ BẢN VỀ BIỂU ĐỒ HỢP TÁC (HTAC) (2)

Collaboration diagram basics

The **collaboration diagram** is part of the Unified Modeling Language (UML). It is one of the two interaction diagrams, the other being the sequence diagram.

Collaboration diagrams convey the same information as sequence diagrams. The main difference is that the collaboration diagram focuses on objects in action, it shows a network of objects that are collaborating (focusing on object structure) whereas the sequence diagram focuses on interactions based on chronology.

Why build a collaboration diagram?

A collaboration diagram represents behavior in terms of interactions. It complements the class diagram that represents the static structure of the system by specifying the behavior of classes, interfaces, and the possible use of their operations.

5.3A.1. CƠ BẢN VỀ BIỂU ĐỒ HỢP TÁC (HTAC) (3)

Converting a collaboration diagram to a sequence diagram

You can convert a collaboration diagram to a sequence diagram.

When you convert a collaboration diagram to a sequence diagram, the sequence diagram does not contain new objects and messages but it reuses the objects and messages displayed in the original collaboration diagram. The instance links used in the collaboration diagram are ignored.

5.3A.2. ĐỊNH NGHĨA CÁC TÁC NHÂN TRONG BIỂU ĐỒ HTAC

Defining actors in a collaboration diagram

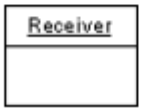
The system you are describing interacts with **actors**. The definition of an actor is the same as in a use case, or a sequence diagram, it characterizes an outside user, or a related set of users that interact with a system.

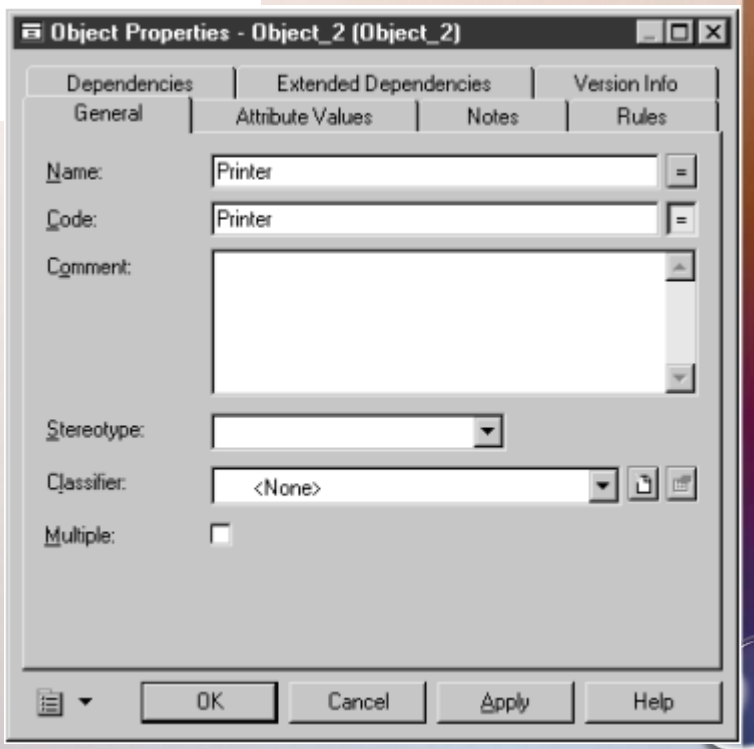
5.3A.3. ĐỊNH NGHĨA CÁC ĐỐI TƯỢNG TRONG BIỂU ĐỒ HTAC

Defining objects in a collaboration diagram

An **object** is an instance of a class. It can be persistent or transient: persistent is the situation of an object that continues to exist after the process that created it has finished, and transient is the situation of an object that stops to exist when the process that created it finishes.

The name of the object is displayed underlined. The Underline character traditionally indicates that an element is an instance of another element.





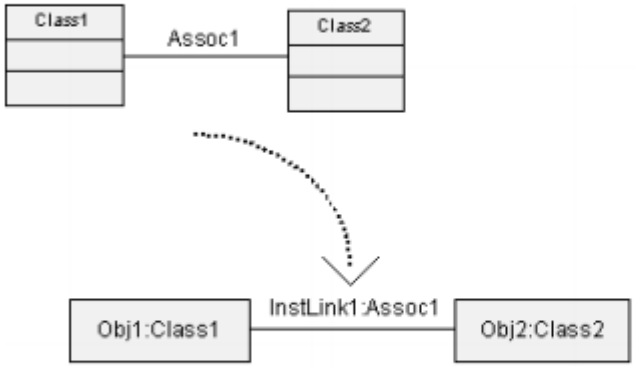
5.3A.4. ĐỊNH NGHĨA CÁC INSTANCE LIÊN KẾT TRONG BIỂU HTAC

Defining instance links in a collaboration diagram

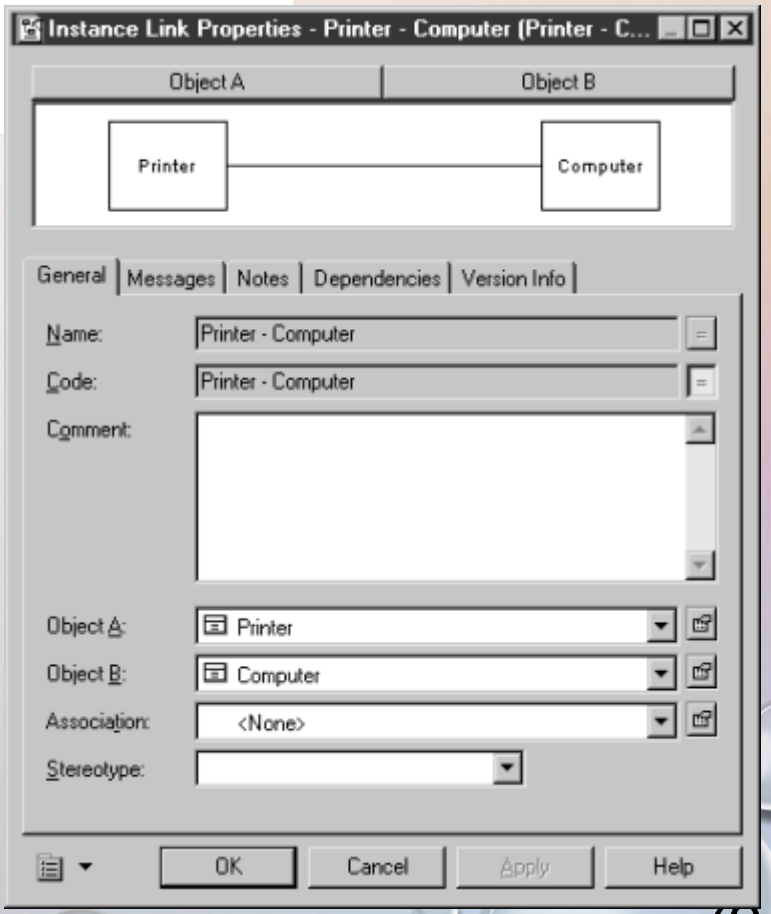
An instance link represents a connection between objects, it highlights the collaboration between objects, hence the name 'collaboration diagram'. It is drawn as a solid line between:

- Two objects
- An object and an actor (and vice versa)

An instance link can be an instance of an association between classes, or an association between a class and an interface.



The instance link in the collaboration diagram shares the same concept as in the object diagram.



5.3A.5. ĐỊNH NGHĨA CÁC MESSAGE TRONG BIỂU ĐỒ HTAC (1)

Defining messages in a collaboration diagram

Objects can cooperate by using several kinds of requests (send a signal, invoke an operation, create an object, delete an existing object, etc.). Sending a signal is used to trigger a reaction from the receiver in an asynchronous way and without a reply. Invoking an operation will apply an operation to an object in a synchronous or asynchronous mode, and may require a reply from the receiver. All these requests constitute **messages**. They correspond to stimulus in the UML language.

How to draw messages?

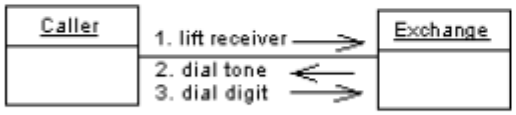
The message symbol is an arrow that displays the following information:

- The direction of the message
- The sequence number
- The message, or the operation name
- The condition
- The argument

A message can be drawn horizontally or vertically in the diagram.

5.3A.5. ĐỊNH NGHĨA CÁC MESSAGE TRONG BIỂU ĐỒ HTAC (2)

Several messages may be attached to the same instance link to specify different interactions. Each message in a collaboration diagram is attached to only one instance link, so the destruction of an instance link destroys all the messages associated with this instance link.

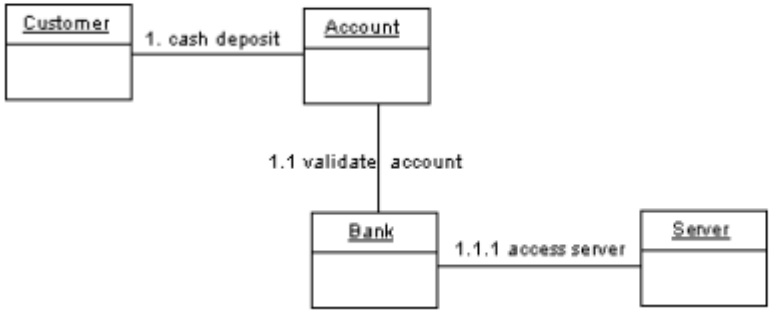


You can create a recursive message: it is a message with the same object as sender and receiver.

Using sequence numbers

A sequence number is appended to a message, it indicates the order in which messages are exchanged. It is the element that defines the order of messages within a collaboration diagram. Sequence numbers are automatically generated if you create the message using the creation tools.

You can use sub-numbers like 1.1, 1.1.1, etc. when you want to describe a nested procedure, or a detailed decomposition of messages exchanged between objects.



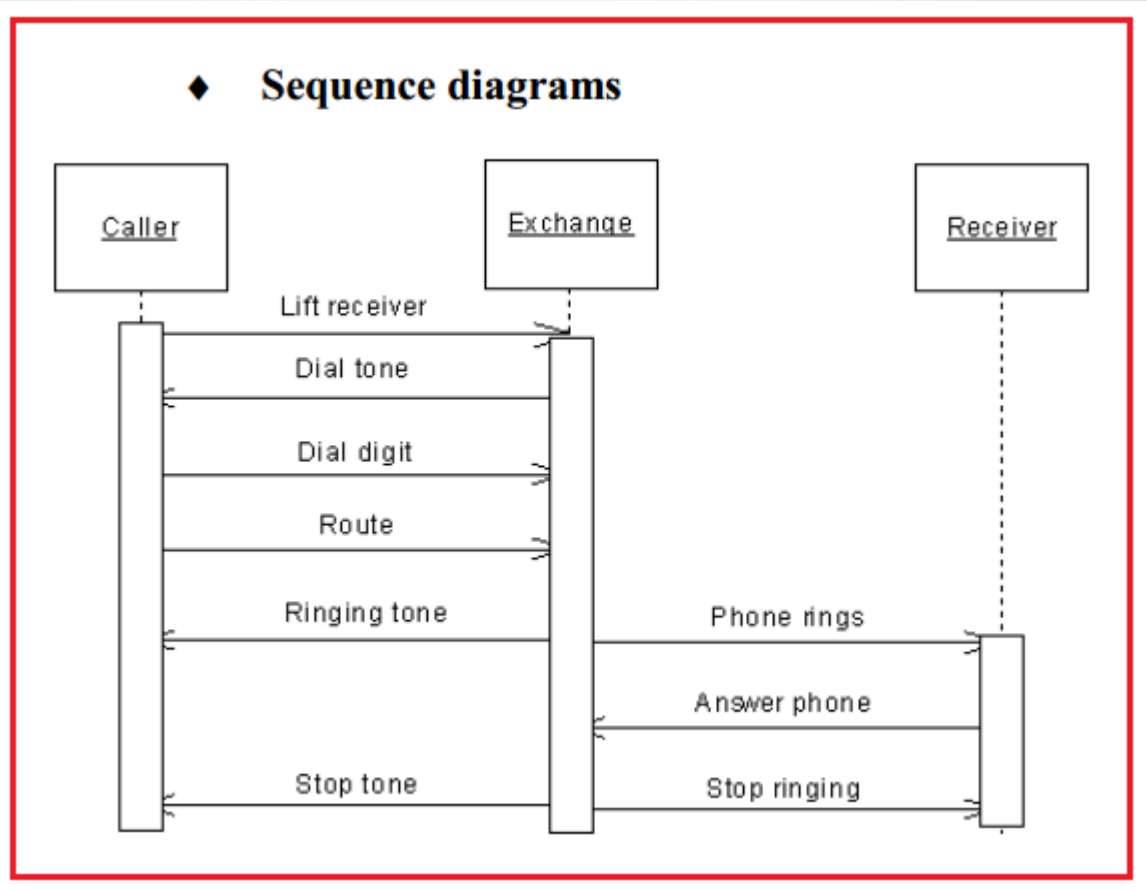
5.3B. XÂY DỰNG BIỂU ĐỒ TUẦN TỰ

5.3B.1. Cơ bản về biểu đồ TUẦN TỰ (TTU)

5.3B.2. Định nghĩa các TÁC NHÂN trong biểu đồ TTU

5.3B.3. Định nghĩa các ĐỐI TƯỢNG trong biểu đồ TTU

5.3B.4. Định nghĩa các MESSAGE trong biểu đồ TTU



5.3B.1. CƠ BẢN VỀ BIỂU ĐỒ TUẦN TỰ (TTU) (1)

Sequence diagram basics

The **sequence diagram** is part of the Unified Modeling Language (UML). It is a graphical description of the operations of a system based on chronology. It represents the life of objects over a definite period of time.

Defining a sequence diagram

The sequence diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It is a dynamic view that shows the symbols of objects (instances of a class), and messages passed along. You can also include actors that interact with the system.

It displays an interaction as a two-dimensional chart. The vertical dimension is the time axis, and the horizontal dimension shows the objects roles that represent individual objects in the collaboration.

Converting a sequence diagram to a collaboration diagram

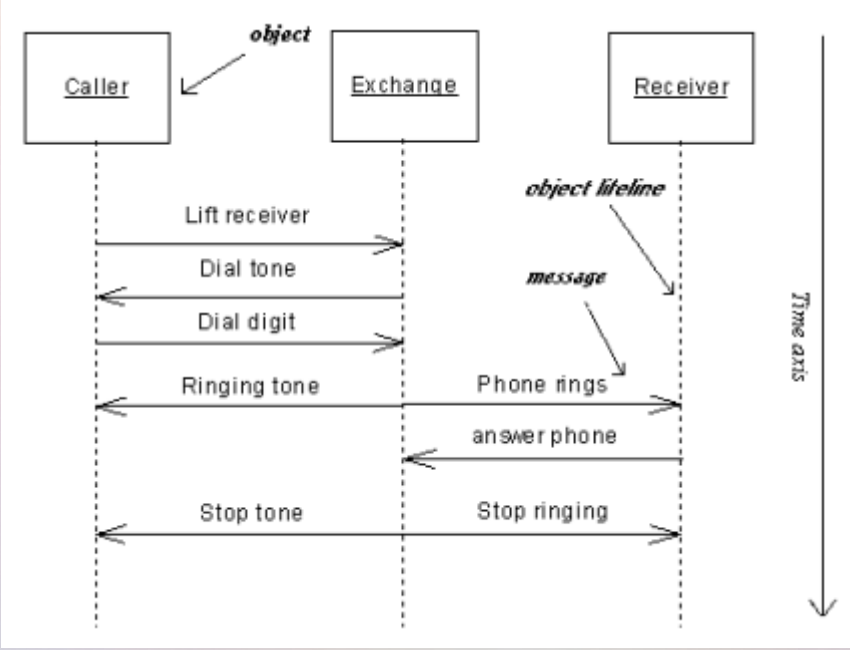
You can convert a sequence diagram to a collaboration diagram.

After you convert a sequence diagram to a collaboration diagram, the collaboration diagram does not contain new objects and messages but it reuses the objects and messages displayed in the original sequence diagram. In addition, instance links are created between the objects that communicate using messages.

4.3B.1. CƠ BẢN VỀ BIỂU ĐỒ TUẦN TỰ (TTU)

Why build a sequence diagram?

A sequence diagram represents behavior in terms of interactions. It complements the class diagram that represents the static structure of the system. It specifies the behavior of classes, interfaces, and the possible use of their operations.



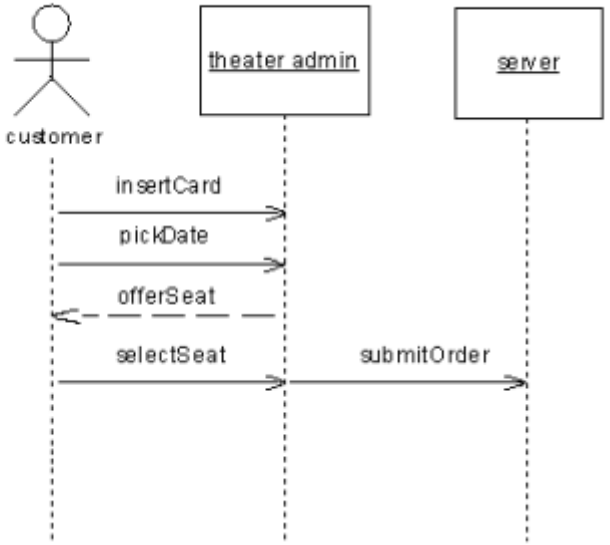
4.3B.2. ĐỊNH NGHĨA CÁC TÁC NHÂN TRONG BIỂU ĐỒ TTU

Defining actors in a sequence diagram

The system you are describing interacts with **actors**. The definition of an actor is the same as in a use case diagram or a collaboration diagram, it characterizes an outside user, or a related set of users that interact with a system.

In the sequence diagram, an actor has a lifeline representing the duration of its life. You cannot separate an actor and its lifeline.

Example



5.3B.3. ĐỊNH NGHĨA CÁC ĐỐI TƯỢNG TRONG BIỂU ĐỒ TTU

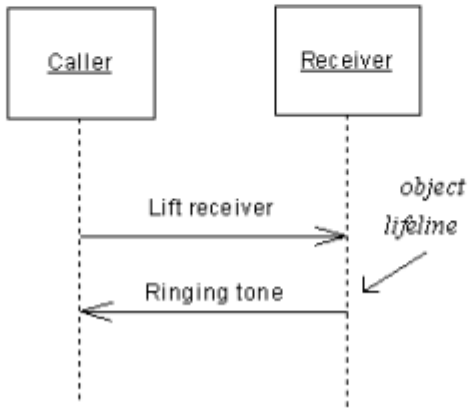
Defining objects in a sequence diagram

An **object** is an instance of a class. It shares the same concept in the object, collaboration and sequence diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

Graphical interpretation

Objects appear at the top of the diagram. They exchange messages between them.

An object that exists when a transaction, or message starts, is shown at the top of the diagram, above the first message arrow. The lifeline of an object that still exists when the transaction is over, continues beyond the final message arrow.



5.3B.4. ĐỊNH NGHĨA CÁC MESSAGE TRONG TTU (1)

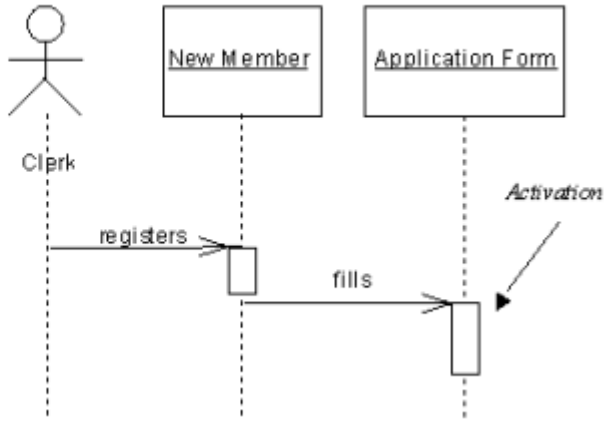
Defining messages in a sequence diagram

A **message** is a communication between objects that conveys information with the expectation that activity will ensue. The receipt of a message should normally have an outcome. It corresponds to a stimulus in UML.

A message has a sender, a receiver, and an action. The **sender** is the object or actor that sends the message. The **receiver** is the object or actor that receives the message. The action is executed to transmit the information.

A message is shown as a horizontal solid arrow from the lifeline of one object or actor, to the lifeline of another object or actor. The arrow is labeled with the name of the message. You can also define a **control flow type** that represents both the relationship between an action and its preceding and succeeding actions, and the waiting semantics between them.

Example

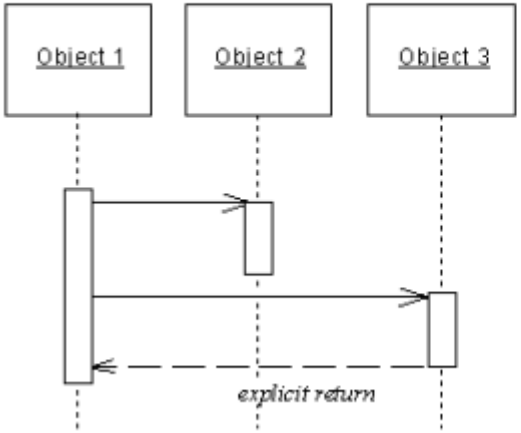


5.3B.4. ĐỊNH NGHĨA CÁC MESSAGE TRONG TTU (2)

A message has the following control flow values:

Control flow	Description	Symbol
Asynchronous	Request in which the sending object does not wait for a result, it can do something else in parallel. **No-wait semantics**	
Procedure Call	Call of a procedure. The sequence is complete before the next sequence resumes. The sender must wait for a response or the end of the activation. **Wait semantics**	
Return	Generally associated with a Procedure Call. The Return arrow may be omitted as it is implicit at the end of an activation	
Undefined	No control flow defined	

In the example below, the explicit Return causes values to be passed back to the original activation.



5.4. XÂY DỰNG BIỂU ĐỒ TRẠNG THÁI, HOẠT ĐỘNG, THÀNH PHẦN, TRIỂN KHAI

5.4A. Xây dựng biểu đồ TRẠNG THÁI

5.4B. Xây dựng biểu đồ HOẠT ĐỘNG

5.4C. Xây dựng biểu đồ THÀNH PHẦN

5.4D. Xây dựng biểu đồ TRIỂN KHAI

5.4A. XÂY DỰNG BIỂU ĐỒ TRẠNG THÁI

5.4A.1. Cơ bản về biểu đồ TRẠNG THÁI (TTHAI)

5.4A.2. Định nghĩa các START trong biểu đồ TTHAI

5.4A.3. Định nghĩa các STATE trong biểu đồ TTHAI

5.4A.4. Định nghĩa các ACTION trong biểu TTHAI

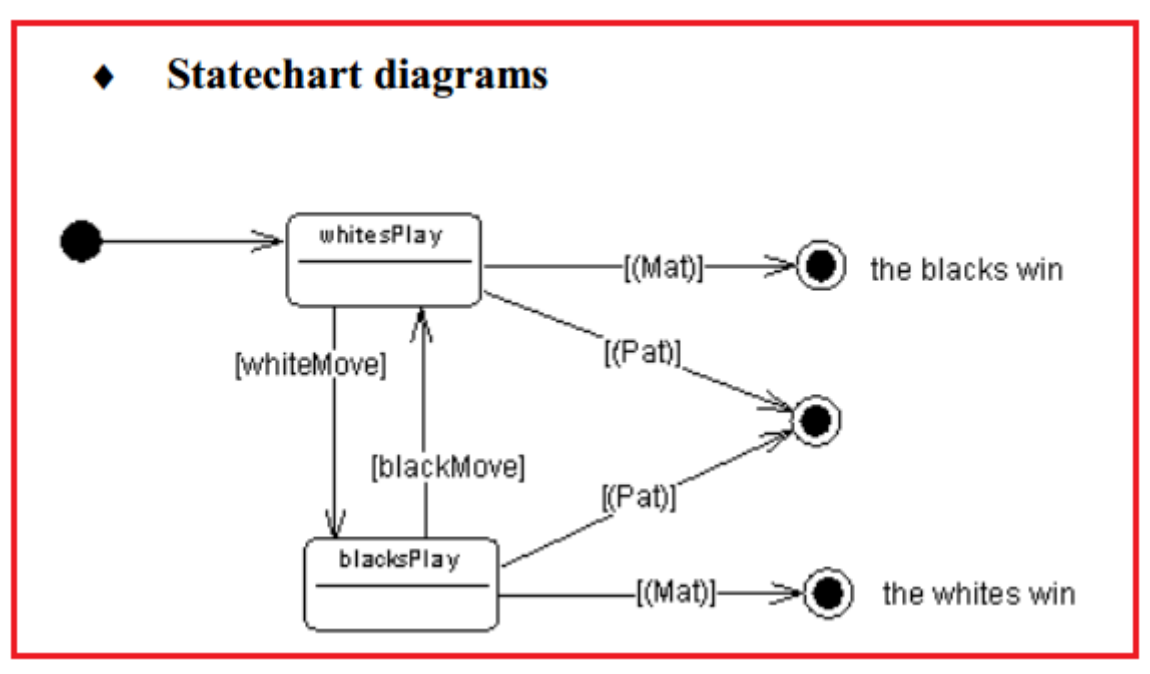
5.4A.5. Định nghĩa các TRANSITION trong biểu đồ TTHAI

5.4A.6. Định nghĩa các EVENT trong biểu đồ TTHAI

5.4A.7. Định nghĩa các FUNCTION POINT trong biểu đồ TTHAI

5.4A.8. Định nghĩa các END trong biểu đồ TTHAI

5.4A. XÂY DỰNG BIỂU ĐỒ TRẠNG THÁI



5.4A.1. CƠ BẢN VỀ BIỂU ĐỒ TRẠNG THÁI (TTHAI) (1)

Statechart diagram basics

The **statechart diagram** is a diagram that describes a classifier behavior.

Defining a statechart diagram

The statechart diagram is one of the diagrams of the Unified Modeling Language (UML). It belongs to the behavioral package in UML (like the use case, collaboration, sequence, and activity diagrams). Its objective is to describe the behavior of a classifier.

5.4A.1. CƠ BẢN VỀ BIỂU ĐỒ TTHAI (2)

Statechart diagram basics

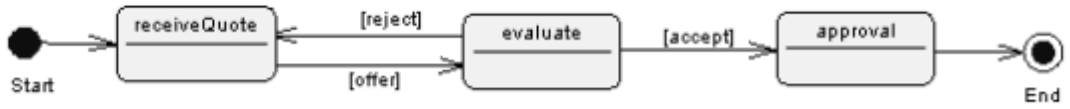
The **statechart diagram** is a diagram that describes a classifier behavior.

Statechart diagram and State machine

The statechart diagram is the graphical representation of a State Machine, a State Machine being a specification that describes the public behavior of some model element that can be a use case, a component or a class. This means that when you start working with a statechart diagram, it is assumed that the classifier has previously been identified in a use case or a class diagram.

The slight difference between a statechart diagram and a State Machine is that the statechart diagram usually represents one instance of a classifier, whereas the State Machine is more global; it represents all instances of the classifier.

The statechart diagram models a finite number of states that the classifier can have, and the events that generate transitions between states.



A statechart diagram can also represent the contents of a composite state, it is called a sub-statechart diagram in this case.

5.4A.1. CƠ BẢN VỀ BIỂU ĐỒ TTHAI (3)

Statechart diagram basics

The **statechart diagram** is a diagram that describes a classifier behavior.

Why build a statechart diagram?

The statechart diagram is built to specify the behavior of an already identified structural element. It specifies a classifier behavior through execution rules explaining precisely how actions are executed during transitions between different states; these states correspond to different situations during the life of the classifier.

The Convert Diagram to State dialog box appears.

5.4A.2. ĐỊNH NGHĨA CÁC START TRONG BIỂU ĐỒ TTHAI

Defining starts in a statechart diagram

A **start** is a starting point of the whole process represented in the statechart diagram. Its symbol is a solid ball as shown below:

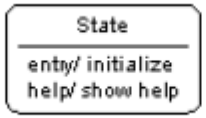
The start has the same meaning in the statechart diagram and in the activity diagram, that is why you can use the same start simultaneously in a statechart and in an activity diagram.

5.4A.3. ĐỊNH NGHĨA CÁC STATE TRONG BIỂU ĐỒ TTHAI

Defining states in a statechart diagram

A **state** represents a situation during the life of a classifier that is usually specified by conditions. It can also be defined as the situation of a classifier waiting for events. Stability and duration are two characteristics of a state. Several states in a statechart diagram correspond to several situations during the life of the classifier.

Example



A state can be atomic or composite:

- An atomic state does not contain sub-states
- A composite state uses sub-states to describe its actions

5.4A.4. ĐỊNH NGHĨA CÁC ACTION TRONG BIỂU TTHAI

Defining actions in a statechart diagram

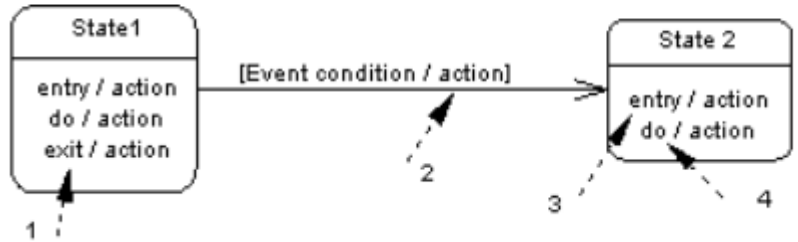
An **action** is a specification of a computable statement. It occurs in a specific situation and may comprise predefined events (entry, do and exit) and internal transitions.

An action contains a **Trigger Event** property containing the specification of the event that triggers the action.

5.4A.4. ĐỊNH NGHĨA CÁC ACTION TRONG BIỂU TTHAI

Action on state and on transition	In an OOM, an action is used in the statechart diagram in association with states: the action is executed in the state during entry or exit. It is also used in association with transitions: the action is executed when the transition is triggered.

In the following figure, you can see actions defined on states, and actions defined on transitions together with the order of execution of actions:

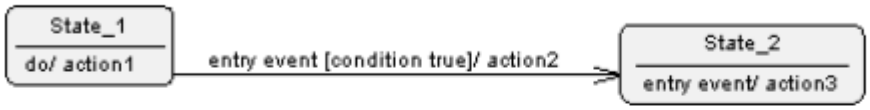


5.4A.5. ĐỊNH NGHĨA CÁC TRANSITION TRONG BIỂU ĐỒ TTHAI (1)

Defining transitions in a statechart diagram

A **transition** is an oriented link between states indicating that an element in a state can enter another state when an event occurs (and if a guard condition is satisfied, when there is one). The expression commonly used in this case is that a transition is fired.

The transition link is represented as a simple line with a direction: an arrow. It contains information displayed above its symbol: the associated event, the condition and the action to execute.



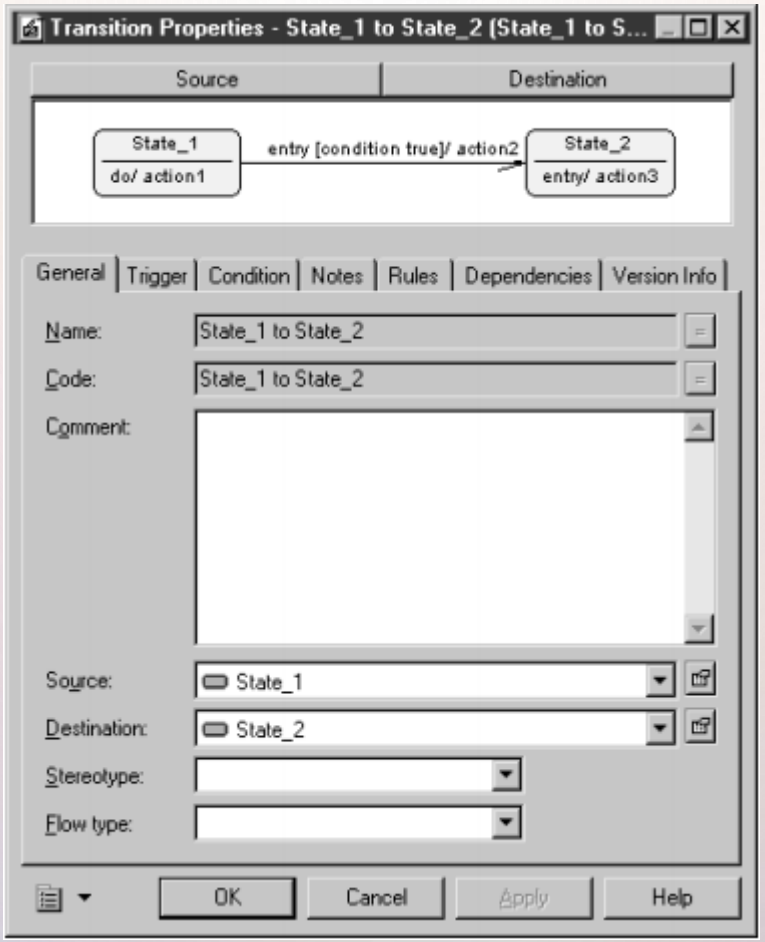
You can draw a transition from and to the following objects:

From\to	Start	State	Junction point	End
Start	—	✔	✔	—
State	—	✔	✔	✔
Junction point	—	✔	✔	✔
End	—	—	—	—

✔ = allowed

— = not allowed

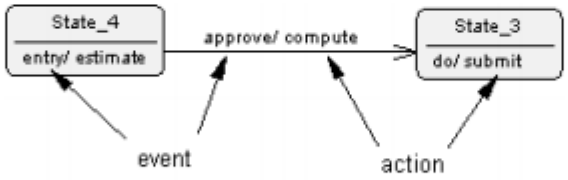
5.4A.5. ĐỊNH NGHĨA CÁC TRANSITION TRONG BIỂU ĐỒ TTHAI (2)



5.4A.6. ĐỊNH NGHĨA CÁC EVENT TRONG BIỂU ĐỒ TTHAI

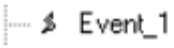
Defining events in a statechart diagram

An **event** is the occurrence of something observable. The occurrence is assumed to be instantaneous and should not have duration. Events convey information specified by parameters. They are used in the statechart diagram in association with transitions: they are attached to transitions to specify which event fires the transition. They are also used in association with actions: the event can trigger the change of state of a classifier or the execution of an internal action on a state.



The same event can be shared between several transitions and actions. It is reusable by nature because it is not dependent on the context.

The event icon in the Browser is the following symbol:



Examples

Here are some examples of what an event can be:

- A boolean expression becoming true
- The reception of a signal
- The invocation of an operation
- A time event, like a timeout or a date reached

You can display the arguments of an event in the statechart diagram.

5.4A.7. ĐỊNH NGHĨA CÁC FUNCTION POINT TRONG BIỂU ĐỒ TTHAI

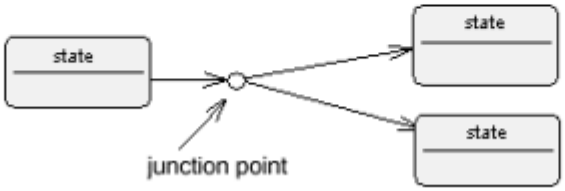
Defining junction points in a statechart diagram

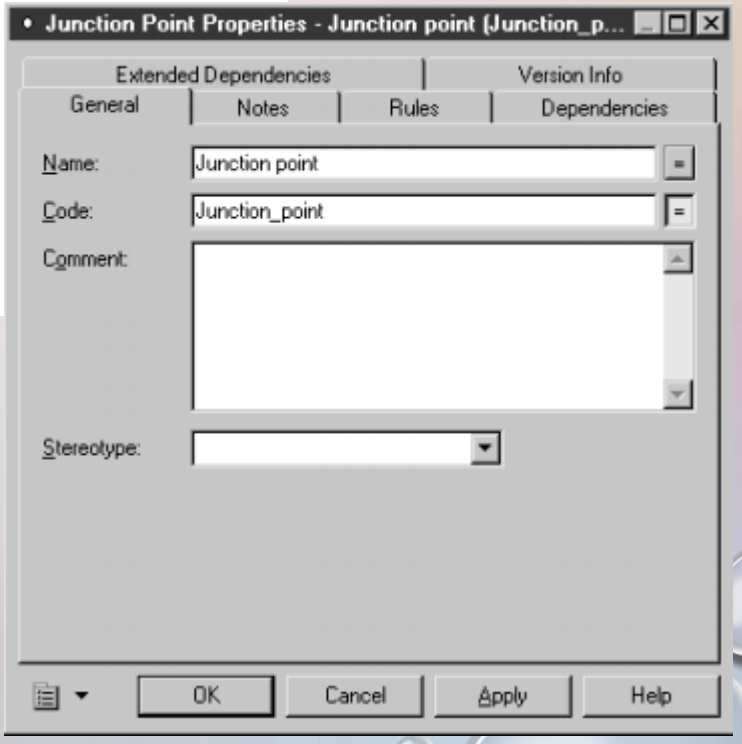
A **junction point** is similar to the decision in the activity diagram, except that it accepts several input and output transitions. Junction points are used to merge and split several transition paths in a statechart diagram.

You are not allowed to use shortcuts of a junction point. A junction point may be dependent on event parameters if the parameters include some split or merge variables for example.

You can attach two transitions of opposite directions to the same junction point symbol.

The symbol of a junction point is an empty circle:



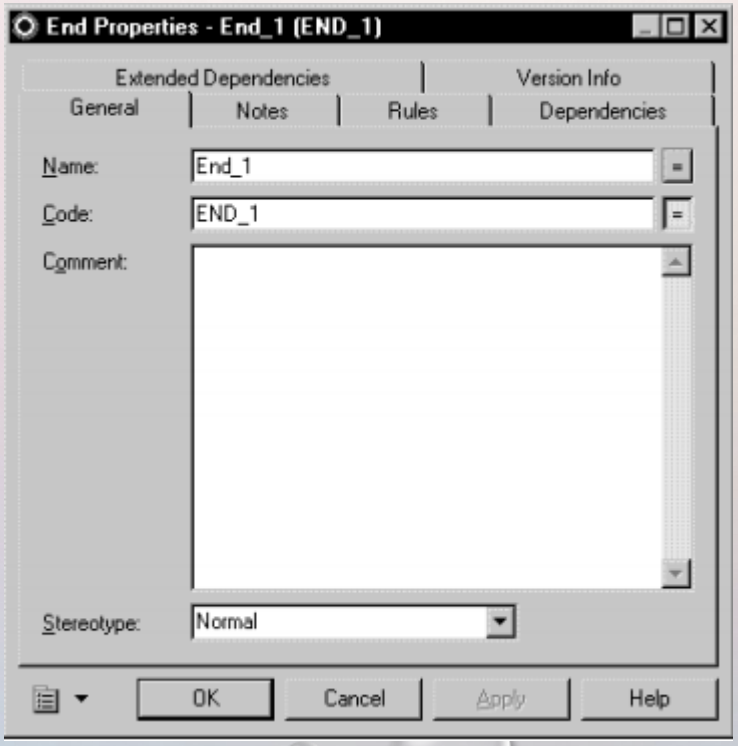


5.4A.8. ĐỊNH NGHĨA CÁC END TRONG BIỂU ĐỒ TTHAI

Defining ends in a statechart diagram

An **end** is a termination point of the states described in the statechart diagram. Its symbol is a solid ball inside a circle (bull's eye).

There can be several ends in the same statechart diagram if you want to show divergent end cases, like errors scenarios. There can also be no end at all if you want to show an endless process.



5.4B. XÂY DỰNG BIỂU ĐỒ HOẠT ĐỘNG

5.4B.1. Cơ bản về biểu đồ HOẠT ĐỘNG (HDONG)

5.4B.2. Định nghĩa các START trong biểu đồ HDONG

5.4B.3. Định nghĩa các ACTIVITY trong biểu đồ HDONG

5.4B.4. Định nghĩa các OBJECT STATE trong biểu HDONG

5.4B.5. Định nghĩa các ORG. UNIT trong biểu đồ HDONG

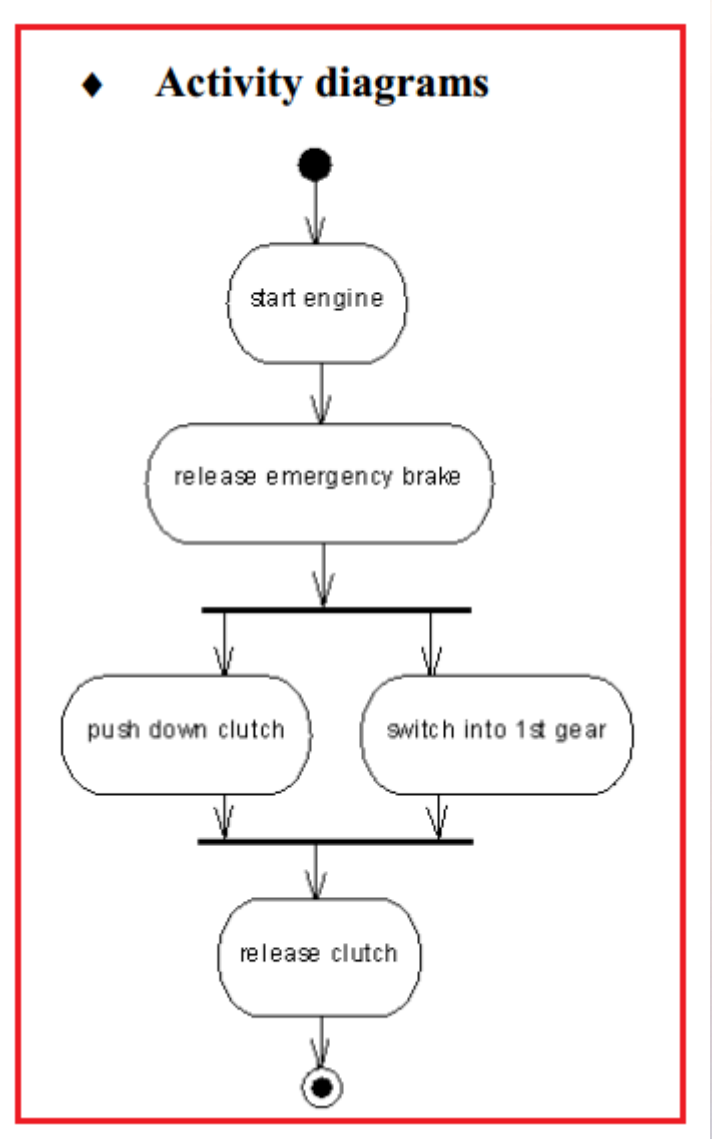
5.4B.6. Định nghĩa các TRANSITION trong biểu đồ HDONG

5.4B.7. Định nghĩa các DECISION trong biểu đồ HDONG

5.4B.8. Định nghĩa các SYNCHRONIZATION trong biểu đồ HDONG

5.4B.9. Định nghĩa các END trong biểu đồ HDONG

5.4B. XÂY DỰNG BIỂU ĐỒ HOẠT ĐỘNG



5.4B.1. CƠ BẢN VỀ BIỂU ĐỒ HOẠT ĐỘNG (HDONG) (1)

Activity diagram basics

The **activity diagram** is a diagram representing a system behavior, particularly suitable for analysis or documentation purposes. It shows control flows (called transitions) between actions performed in the system (called activities) from a start point to several potential end points.

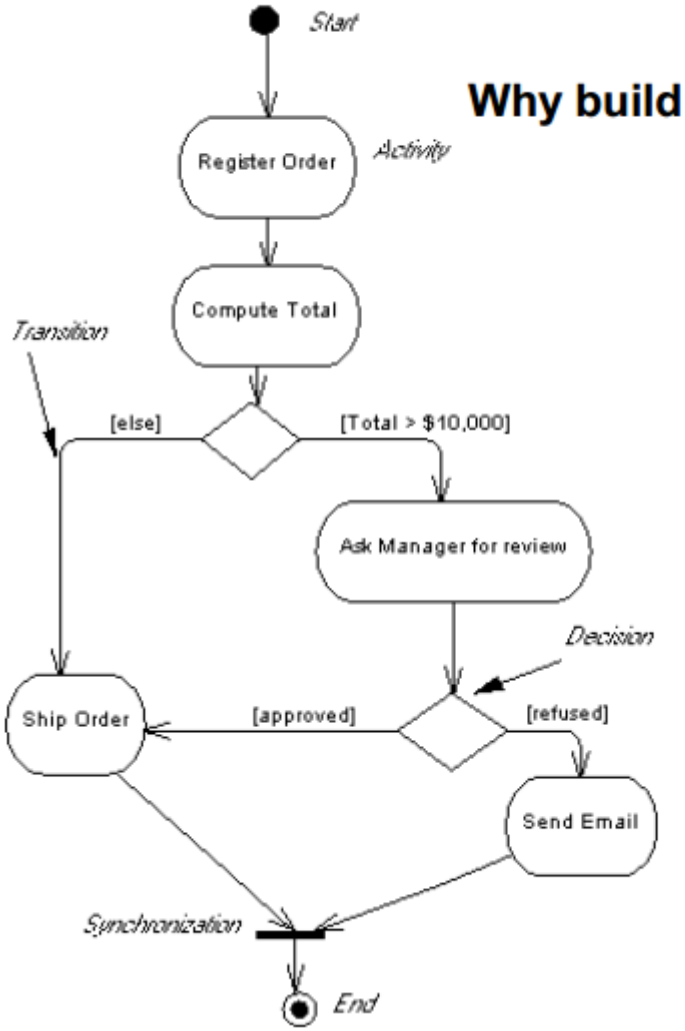
Defining an activity diagram

The activity diagram is one of the diagrams of the Unified Modeling Language (UML). It is a simplification of the UML statechart diagram for modeling control flows in computational and organizational processes.

The activity diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of the objects defined in the model. The main objects are the activities and the transitions between them.

5.4B.1. CƠ BẢN VỀ BIỂU ĐỒ HDONG (2)

An activity diagram focuses on the operations that are passed among activities.



Why build an activity diagram?

An activity diagram is used to describe processes in which activities represent the completion of internally generated actions. It is meant to represent the internal behavior of a method (implementation of an operation), a classifier or a use case.

5.4B.2. ĐỊNH NGHĨA CÁC START TRONG BIỂU ĐỒ HDONG

Defining starts in an activity diagram

A **start** is a starting point of the whole process represented in the activity diagram. Its symbol is a solid ball as shown below:

5.4B.3. ĐỊNH NGHĨA CÁC ACTIVITY TRONG BIỂU ĐỒ HDONG (1)

Defining activities in an activity diagram

An **activity** is the invocation of a manual or automated action. When the activity gains the control, it performs the action, then, depending on the result of the action, the transition (control flow) is passed to another activity.

An activity can be atomic or composite:

- An atomic activity does not contain sub-activities
- A composite activity uses sub-activities to describe its actions

Example

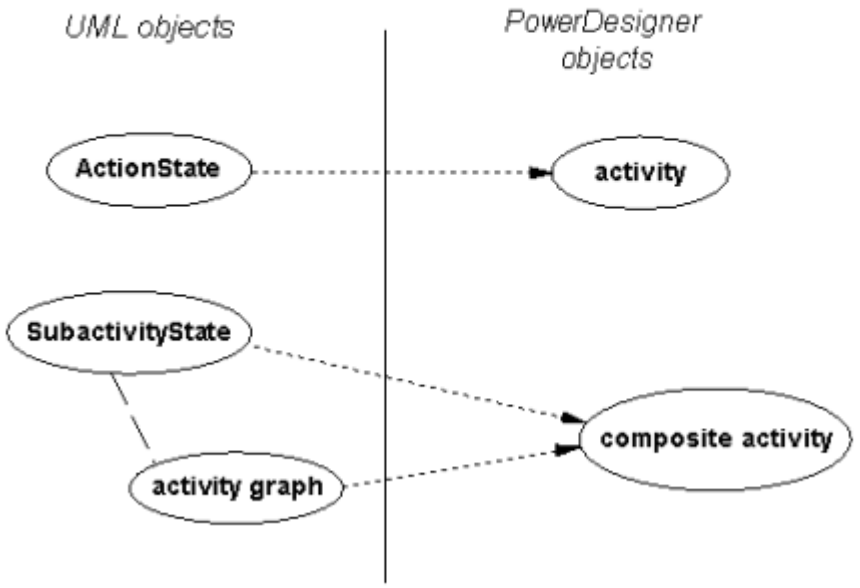
An activity may be the invocation of an action like "send a mail", or "increment a counter".

The symbol of an activity is the following:

You can use internal and external shortcuts of activities in your model.

5.4B.3. ĐỊNH NGHĨA CÁC ACTIVITY TRONG BIỂU ĐỒ HDONG (2)

The following chart highlights UML vs PowerDesigner terminology and concepts:



5.4B.4. ĐỊNH NGHĨA CÁC OBJECT STATE TRONG BIỂU HDONG (1)

Defining object states in an activity diagram

An **object state** is the association of an object (instance of a class) and a state. It represents an object in a particular state.

Its symbol is a rectangle as shown below:

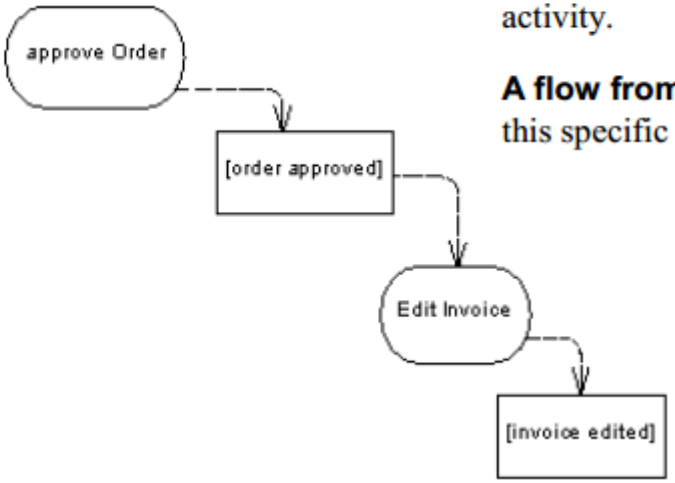
In the activity diagram, the same object can evolve after several actions defined by activities, have been executed. For example, a document can evolve from the state initial, to draft, to reviewed, and finally turn into a state approved.

You can draw a link from an activity to an object state and inversely:

A flow from an activity to an object state means that the execution of the activity puts the object in a specific state. It represents the result of an activity.

A flow from an object state to an activity means that the activity uses this specific state in its execution. It represents a data flow between them.

Example

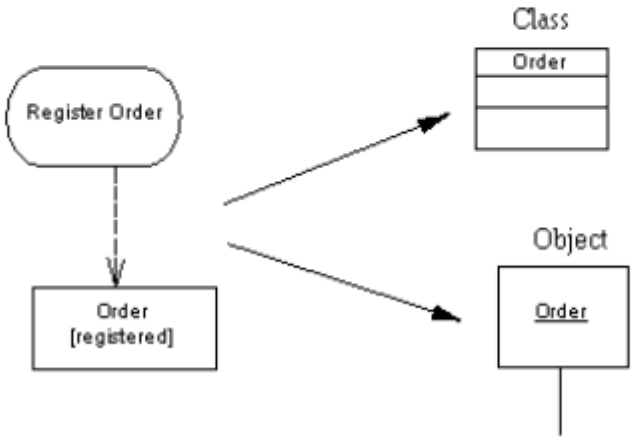


5.4B.4. ĐỊNH NGHĨA CÁC OBJECT STATE TRONG BIỂU HDONG (2)

Global example

How are object states involved in the workflow associated with an activity diagram?

For example, an object state whose state is **[registered]** can be associated with an object **Order**, this object can emerge from the sequence diagram from some activity named **Register Order**. Furthermore this same object can be related to the class Order represented in a class diagram in the model.



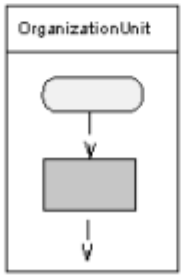
5.4B.5. ĐỊNH NGHĨA CÁC ORG. UNIT TRONG BIỂU ĐỒ HDONG

Defining organization units in an activity diagram

An **organization unit** is an optional element that allows you to graphically highlight which organization unit is responsible for which activity. It can represent a company, a system, a service, an organization, a user or a role.

The organization unit is equivalent to the **swimlane** in UML. In the OOM, the organization unit is the tangible element and the swimlane is the symbol that represents the organization unit.

Verify that the Organization Unit Swimlane check box is selected from the Tools→Display Preferences→General command. The graphical symbol of the organization unit will then be a swimlane. It can contain all the symbols of an activity diagram: activities, object states, transitions, etc. as shown below:



5.4B.6. ĐỊNH NGHĨA CÁC TRANSITION TRONG BIỂU ĐỒ HDONG (1)

Defining transitions in an activity diagram

The transition in the activity diagram shares the same concept as in the statechart diagram.

Transition between activities	A **transition** is a link between two activities represented as a line going from one activity to another. It is a route the control flow transits on to link activities. The routing of the control flow is made using guard conditions defined on transitions. If the condition is true, the control is passed to the next element. The transition link is represented as a simple line with a direction (arrow). 
Transition between activity and object state	A transition between an activity and an object state indicates that the execution of an activity puts an object in a specific state. When a specific event occurs or when specific conditions are satisfied, the control flow passes from the activity to the object state. Inversely, a transition between an object state and an activity means that the activity uses this specific state in its execution. The transition link is represented as a dashed line with a direction (arrow). 

5.4B.6. ĐỊNH NGHĨA CÁC TRANSITION TRONG BIỂU ĐỒ HDONG (2)

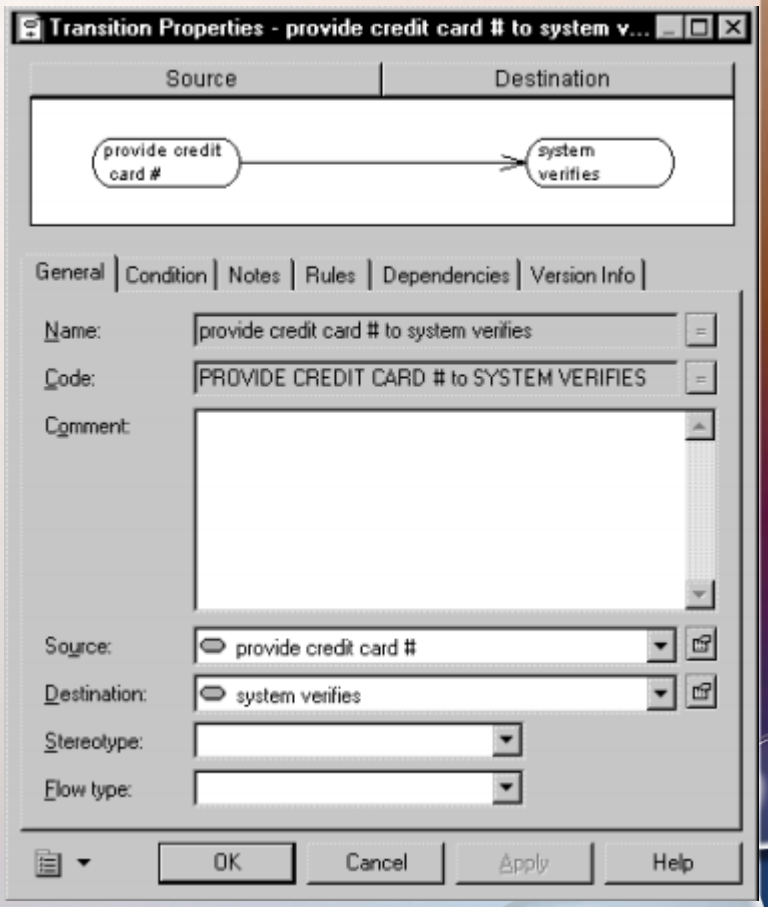
In addition to the above mentioned transitions, you can draw a transition from and to the following objects:

From\to	Start	Activity	OS*	Decision	Fork	Join	End
Start	—	✔	✔	✔	✔	—	—
Activity	—	✔	✔	✔	✔	✔	✔
OS*	—	✔	—	✔	✔	✔	—
Decision	—	✔	✔	✔	✔	✔	✔
Fork	—	✔	✔	✔	✔	✔	—
Join	—	✔	✔	✔	✔	✔	✔
End	—	—	—	—	—	—	—

✔ = allowed

— = not allowed

***OS** = Object State



5.4B.7. ĐỊNH NGHĨA CÁC DECISION TRONG BIỂU ĐỒ HDONG (1)

Defining decisions in an activity diagram

A **decision** specifies which alternate path has to be taken when several transition paths are possible. It can have one or more input transitions and one or more output transitions, each labeled with a distinct guard condition. The process of a decision is based on some defined expressions that direct the control flow towards the valid transition by dynamically evaluating guard conditions.

Applicability

The symbol of a decision is a diamond shape:

A decision allows you to create complex flows like:

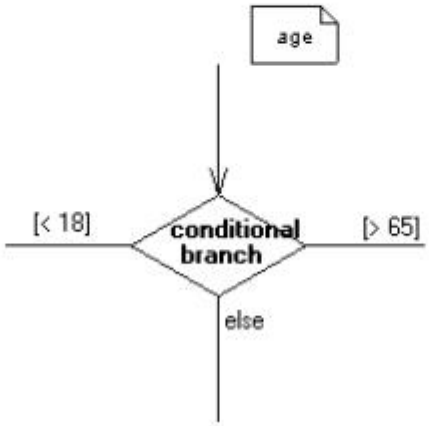
- if … then … else …
- switch … case …
- do … while …
- loop
- for … next …

5.4B.7. ĐỊNH NGHĨA CÁC DECISION TRONG BIỂU ĐỒ HDONG (2)

Defining conditional branches

A **conditional branch** joins multiple transitions. It is a place within an activity diagram where a specific event leads to more than one possible outcome, each of which has an associated guard condition or the keyword ELSE.

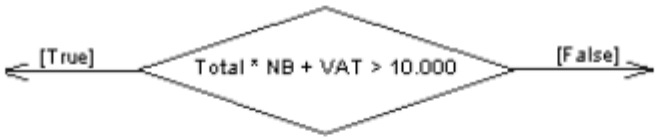
Symbol



Factorizing conditions

If you are working with a conditional branch, it is useful to write a condition on the decision in order to factorize the conditions attached to the transitions, it allows you to simplify the process whenever you are using long and complex conditions.

Example



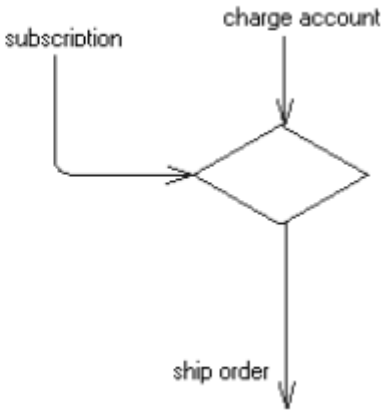
Use the Condition page in the decision property sheet to write the following condition: Total * NB + VAT > 10.000. Then use the Condition page in both transitions property sheet: enter True in one and False in the other.

5.4B.7. ĐỊNH NGHĨA CÁC DECISION TRONG BIỂU ĐỒ HDONG (3)

Defining merges

A **merge** is a scenario within an activity diagram where two or more alternate control paths come together, it represents the merge of several potential transitions into a common path.

Symbol



5.4B.8. ĐỊNH NGHĨA CÁC SYNCHRONIZATION TRONG BIỂU ĐỒ HDONG (1)

Defining synchronizations in an activity diagram

A **synchronization** is an object that enables synchronization of control between two or more concurrent actions. Its symbol is usually a thick horizontal line, like shown below, but it can also be drawn vertically.

When working with a synchronization, you can choose to either use a **fork** or a **join**. They have different layouts, depending on the number of input and output transitions.

5.4B.8. ĐỊNH NGHĨA CÁC SYNCHRONIZATION TRONG BIỂU ĐỒ HDONG (2)

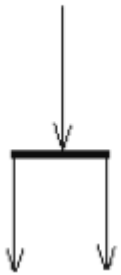
Defining forks

A **fork** is the splitting of an input transition into several output transitions executed in parallel. It creates several concurrent branches below its symbol. The activities associated with each of these paths continue in parallel.

A fork should have only one input flow and have more than one output flow.

It is a complex transition within which one source activity is replaced with two or more destination activities, each of which representing an independent transition (or control flow).

Symbol



5.4B.8. ĐỊNH NGHĨA CÁC SYNCHRONIZATION TRONG BIỂU ĐỒ HDONG (3)

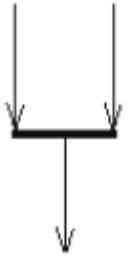
Defining joins

A **join** is the synchronization of several parallel transitions. It is a complex transition with two or more source activities and one destination activity.

A join should have more than one input flow, and have only one output flow.

The activities associated with each of the paths above the join symbol are parallel. In a join, concurrent transitions synchronize. Each transition waits until all input flows reach the join before continuing; the first executed transition waiting for the last one to complete. After the join, only one transition continues below the join.

Symbol

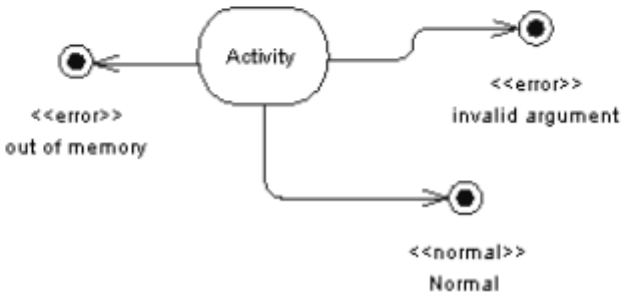


5.4B.9. ĐỊNH NGHĨA CÁC END TRONG BIỂU ĐỒ HDONG

Defining ends in an activity diagram

An **end** is a termination point of the activities described in the activity diagram. Its symbol is a solid ball inside a circle (bull's eye).

There can be several ends in the same activity diagram if you want to show divergent end cases, like errors scenarios for example:



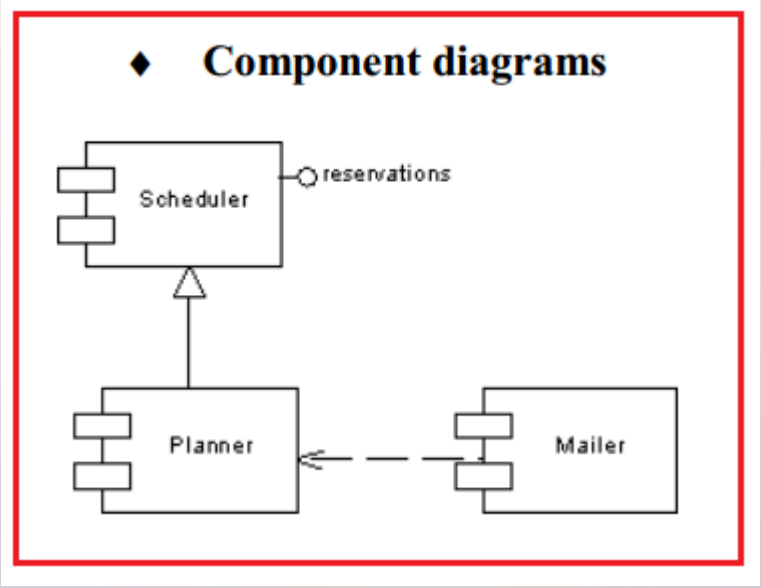
5.4C. XÂY DỰNG BIỂU ĐỒ THÀNH PHẦN

5.4C.1. Cơ bản về biểu đồ THÀNH PHẦN (TPHAN)

5.4C.2. Định nghĩa các COMPONENT trong biểu đồ TPHAN

5.4C.3. Định nghĩa các GENERALIZATION trong biểu đồ TPHAN

5.4C.4. Định nghĩa các DEPENDENCY trong biểu TPHAN



5.4C.1. CƠ BẢN VỀ BIỂU ĐỒ THÀNH PHẦN (TPHAN)

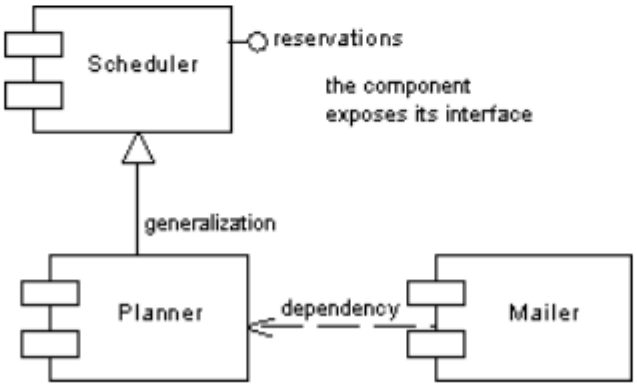
Component diagram basics

The **component diagram** is part of the Unified Modeling Language (UML). It shows dependencies among software components.

Defining a component diagram

The component diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of components defined in the model including interfaces defined in components and the relationships between them.

It addresses the static implementation view of a system, drawn as a graph of software components connected by dependencies. Components may also be connected by generalization relationships when a component inherits from another one.



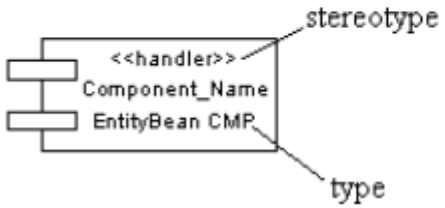
5.4C.2. ĐỊNH NGHĨA CÁC COMPONENT TRONG BIỂU ĐỒ TPHAN

Defining components in a component diagram

A **component** is a physical, replaceable part of a system that packages implementation, conforms to and provides the realization of a set of interfaces. It can represent a physical piece of implementation of a system, like software code (source, binary or executable), scripts, or command files. It is an independent piece of software developed for a specific purpose but not a specific application. It may be built up from the class diagram and written from scratch for the new system, or it may be brought in from other projects and third party vendors.

The symbol of the component is the following:

To display the type of a component, select Tools→Display Preferences and select the Show Type option in the component category.

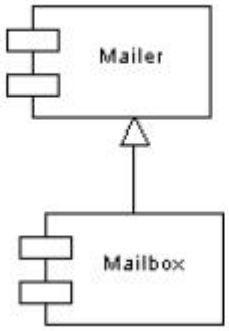


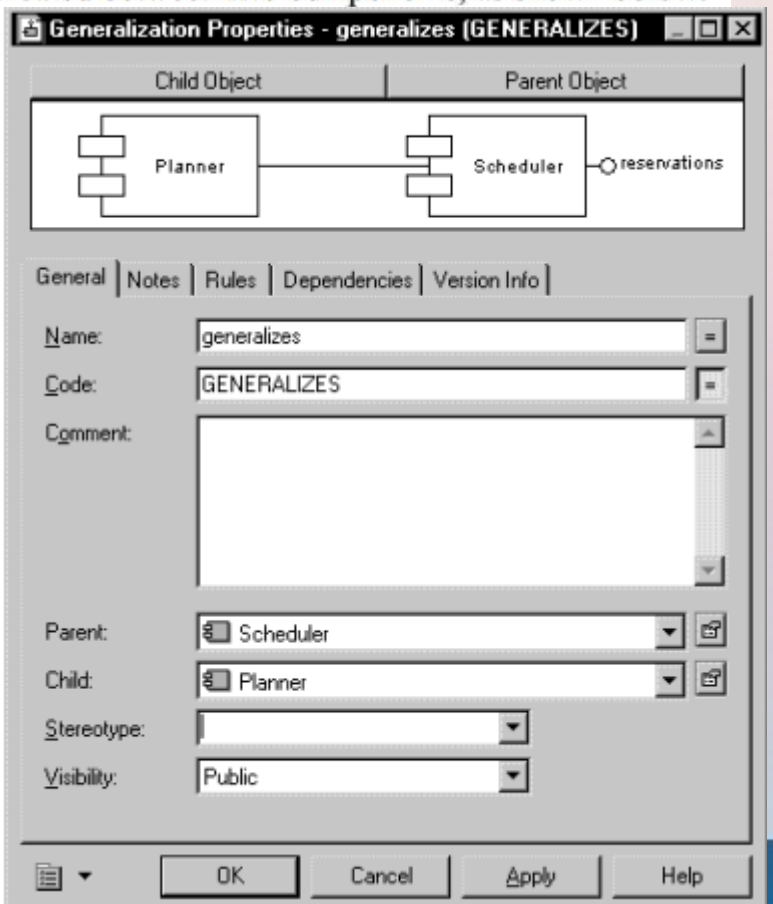
5.4C.3. ĐỊNH NGHĨA CÁC GENERALIZATION TRONG BIỂU ĐỒ TPHAN

Defining generalizations in a component diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

A generalization can be created between two components, as shown below:



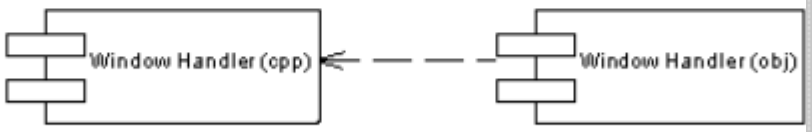


5.4C.4. ĐỊNH NGHĨA CÁC DEPENDENCY TRONG BIỂU TPHAN

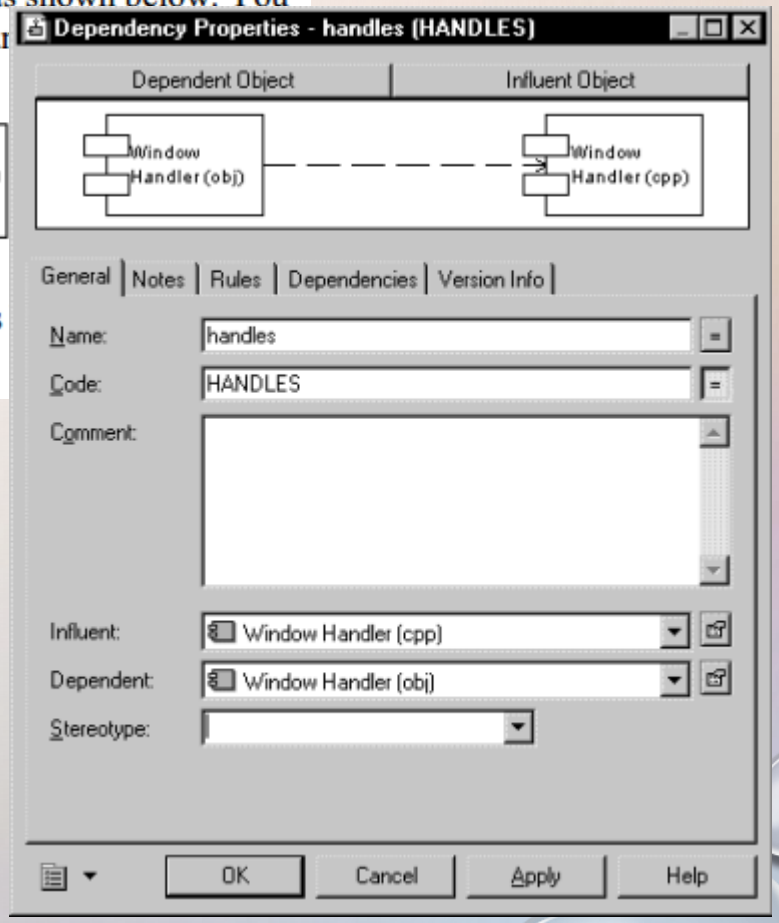
Defining dependencies in a component diagram

A dependency is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

A dependency can be created between two components as shown below. You can **not** create a dependency between a component and ar



When using a dependency, you can nest two components
stereotype.



5.4D. XÂY DỰNG BIỂU ĐỒ TRIỂN KHAI

5.4D.1. Cơ bản về biểu đồ TRIỂN KHAI (TKHAI)

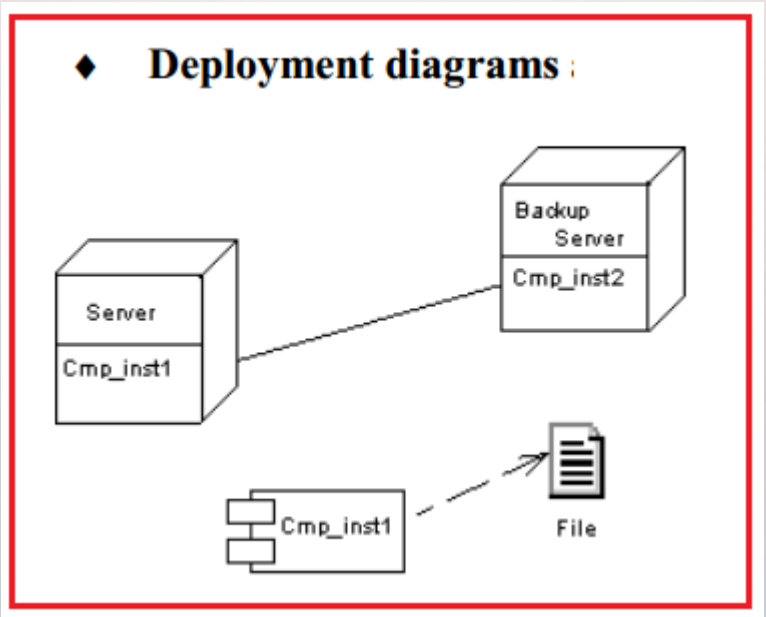
5.4D.2. Định nghĩa các NODE trong biểu đồ TKHAI

5.4D.3. Định nghĩa các CONPONENT INSTANCE trong b.đồ TKHAI

5.4D.4. Định nghĩa các FILE OBJECT trong biểu TKHAI

5.4D.5. Định nghĩa các LIÊN KẾT NODE trong biểu đồ TKHAI

5.4D.6. Định nghĩa các DEPENDENCY trong biểu đồ TKHAI



5.4D.1. CƠ BẢN VỀ BIỂU ĐỒ TRIỂN KHAI (TKHAI) (1)

Deployment diagram basics

The **deployment diagram** shows the physical configuration of run-time processing elements (nodes) on an instance level. The nodes contain instances of component that will be deployed into database, application or web servers.

5.4D.1. CƠ BẢN VỀ BIỂU ĐỒ TRIỂN KHAI (TKHAI) (2)

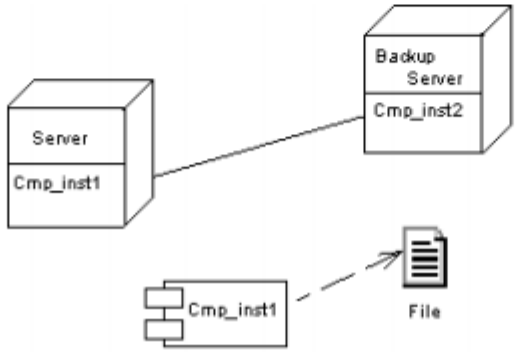
Defining a deployment diagram

The deployment diagram is one of the available diagrams in an OOM (Object-Oriented Model).

The deployment diagram is part of the Unified Modeling Language (UML). It is an implementation diagram that complements the component diagram by giving more accurate details about the physical implementation and interactions of components.

The deployment diagram is a view of nodes connected by communication links. It allows you to design nodes, component instances that can be contained within the node symbol, file objects associated with nodes that are used for deployment, and relationships between nodes.

When the node contains component instances, this indicates that the component executes on the node.

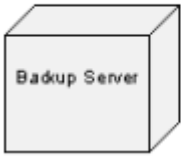


5.4D.2. ĐỊNH NGHĨA CÁC NODE TRONG BIỂU ĐỒ TKHAI (3)

Defining nodes in a deployment diagram

A node is the main element of the deployment diagram. It is a physical element that represents a processing resource, a real physical unit, or physical location of a deployment (computer, printer, or other hardware units).

The symbol of a node is a cube:



5.4D.3. ĐỊNH NGHĨA CÁC CONPONENT INSTANCE TRONG B.ĐỒ TKHAI

Defining component instances in a deployment diagram

A component instance is an instance of a component that can run or execute on a node. Whenever a component is processed into a node, a component instance is created. The component instance plays an important role because it contains the parameter for deployment into a server.

The component instance symbol is the same as the component symbol in the component diagram.

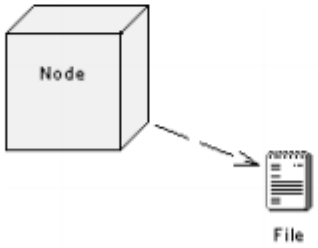
5.4D.4. ĐỊNH NGHĨA CÁC FILE OBJECT TRONG BIỂU TKHAI

Defining file objects in a deployment diagram

A file object can be a bitmap file used for documentation, or it can be a file containing text that is used for deployment into a server.

File_1

When you want to associate a file object to a node, you can drag a dependency from the file object to the node:

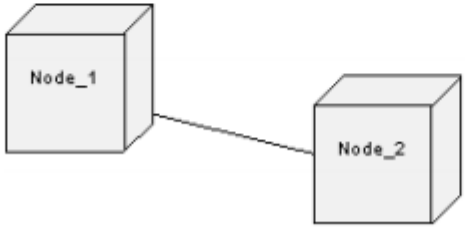


You can also use CTRL and double-click on the parent node symbol, then create the file object into the node diagram.

5.4D.5. ĐỊNH NGHĨA CÁC LIÊN KẾT NODE TRONG BIỂU ĐỒ TKHAI (1)

Defining node associations in a deployment diagram

You can create associations between nodes, called node associations. They are defined with a role name and a multiplicity at each end. An association between two nodes means that the nodes communicate with each other, for example when a server is sending data to a backup server.



5.4D.5. ĐỊNH NGHĨA CÁC LIÊN KẾT NODE TRONG BIỂU ĐỒ TKHAI (2)

Node association multiplicity

The allowable cardinalities of a role are called multiplicity. Multiplicity indicates the minimum and maximum cardinality that a role can have.

Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

The Multiplicity is related to instances of a node association. For example, in a computer environment, there can be 100 clients and 100 machines but there is a constraint that says that a machine can accept at most 4 clients at the same time. In this case, the maximum number of instances is set to 4 in the Multiplicity box on the machine side:

5.4D.6. ĐỊNH NGHĨA CÁC DEPENDENCY TRONG BIỂU ĐỒ TKHAI

Defining dependencies in a deployment diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element). The dependency relationship indicates that one node in a diagram uses the services or facilities of another node.

In a deployment diagram, a dependency can be created between nodes, and component instances.

5.5. LÀM VIỆC VỚI MÔ HÌNH HƯỚNG ĐỐI TƯỢNG

5.5.1. Định nghĩa các biểu đồ related

5.5.2. Liên kết các object

5.5.3. Kiểm tra một OOM

5.5.4. Các đối tượn với mô hình Check

5.5.5. Mapping objects trong OOM

5.5.6. Opening một biểu đồ Rose trong OOM

5.5.1. ĐỊNH NGHĨA CÁC BIỂU ĐỒ RELATED

Defining related diagrams

Each diagram of the OOM contains specific objects. However, when you are closer to implementation, if you want to attach existing objects to other types of diagrams (other than the one they were created in), you can use the **Related Diagrams** feature.

Using related diagrams

Using related diagrams helps you explain in detail how some of the following OOM objects can be viewed from different angles.

5.5.2. LIÊN KẾT CÁC OBJECT (1)

Linking objects to other objects

It is possible to link one object of a diagram type to another object of a different diagram type. You do that to show that one object evolves, through implementation or instantiation into another object. In this case, the evolution of the object is visible in two, or more diagrams of the model. It is also possible to attach objects between different Object-Oriented models.

You can attach an object to another object for the following reasons:

- Follow the projection of the object among different types of diagrams
- Show its implementation into one or more objects
- Show its instantiation into another object
- For documentation purposes also, you may want to follow up the behavior of an object to better identify its relationships with other objects, this may be helpful if you need to resume analysis of your system later.

5.5.2. LIÊN KẾT CÁC OBJECT (2)

You can attach objects in the following manner:

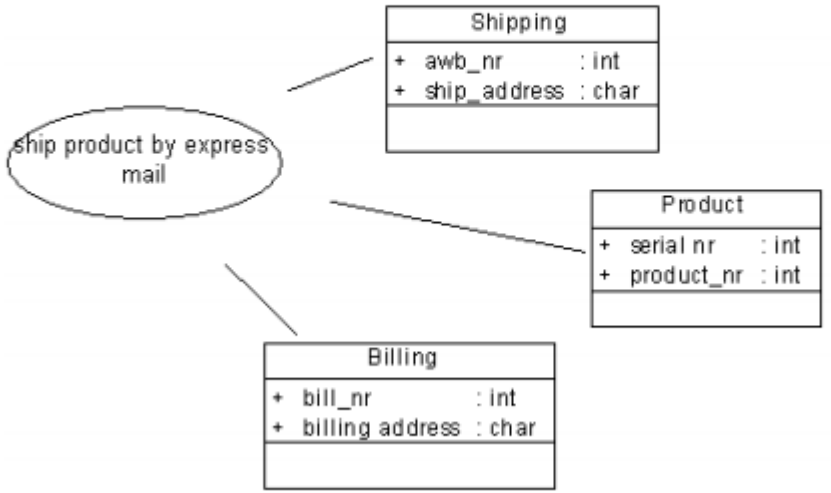
Object	Attached to	See section
Class and interface	Use case and actor	Linking a class or interface to a use case and Linking a class or interface to an actor
Use case, class and component	State	Linking a classifier to a state
Class and interface	Object	Linking a class or interface to an object
Component	Component instance	Linking a component to a component instance
Class	Object state	Linking a class to an object state
Object	Object state	Linking an object to an object state
Message	Operation (class)	Linking a message to an operation
Association	Instance link	Linking an association to an instance link

5.5.2. LIÊN KẾT CÁC OBJECT (3)

Linking a class or interface to a use case

A use case can be used and expanded in class diagrams to describe how the use case is implemented by one or several classes, or interfaces.

For example, a use case **ship product by express mail** needs the classes **Shipping**, **Product**, and **Billing** to perform its task. They are shown as related to the use case in the figure below.



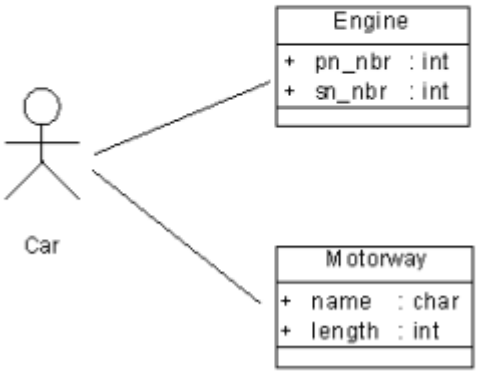
5.5.2. LIÊN KẾT CÁC OBJECT (4)

Linking a class or interface to an actor

During the implementation phase, an actor could be implemented as one or several classes, or one of several interfaces.

For example, an actor Car needs the classes Engine and Motorway to perform its task.

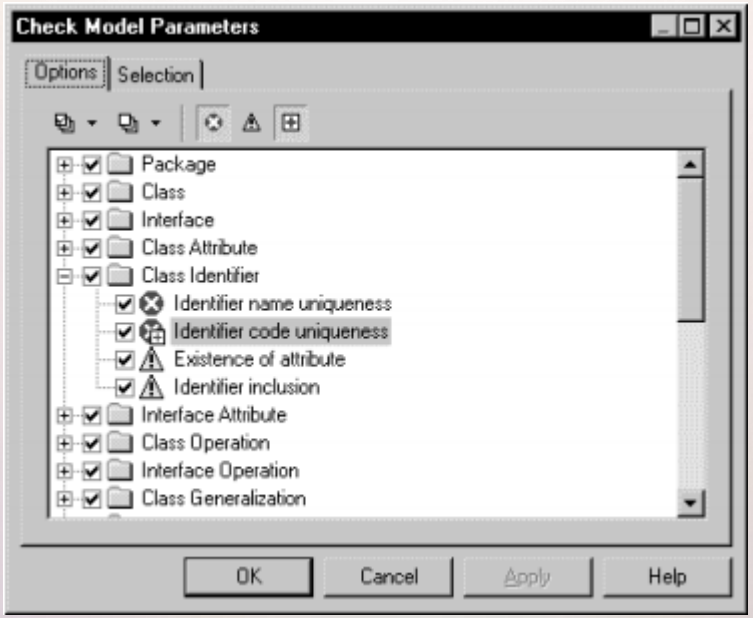
They are shown as related to the actor in the figure below.



5.5.3. KIỂM TRA MỘT OOM

Checking an OOM

You can check the validity of your OOM at any time using the Check Model feature from the Tools menu.



5.5.4. CÁC ĐỐI TƯỢN VỚI MÔ HÌNH CHECK

Objects verified during Check model

The Check Model verifies the validity of the following objects in an OOM.

- Domain check
- Business Rule check in an OOM
- File object check
- Actor check
- Package check
- Use case check
- Class check

5.5.5. MAPPING OBJECTS TRONG OOM

Mapping objects in an OOM

Object mapping is the ability to establish a correspondence between objects belonging to heterogeneous models and diagrams. In an OOM, you can link classes to tables in a PDM, this process is called **object to relational mapping** (O/R mapping).

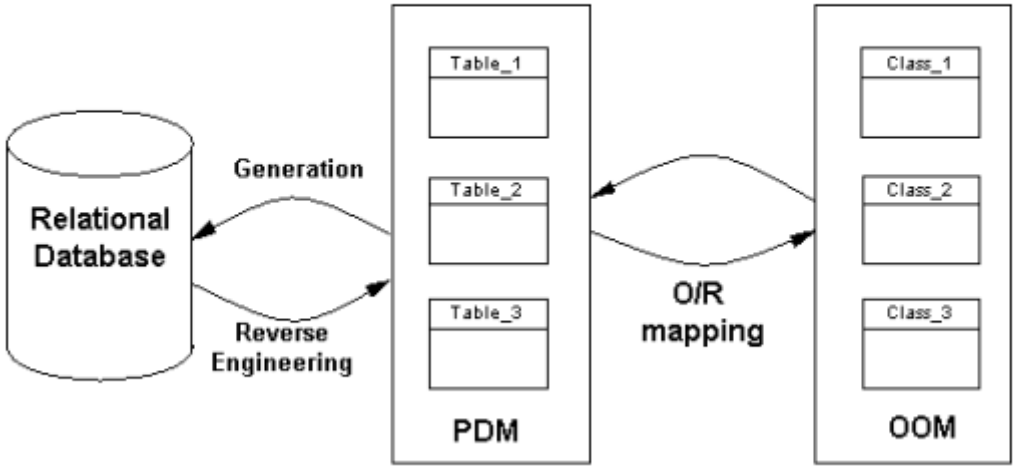
Understanding O/R mapping

You create a mapping between OOM objects and PDM tables to setup object persistence. Data comes from a **data source** and is managed by an OOM. The mapping appears as a query defined in the OOM object: this query allows you to handle data in the data source.

In O/R mapping, the data source contains one or several physical data models representing the structure of the relational database in which OOM objects will be stored.

5.5.5. MAPPING OBJECTS TRONG OOM (1)

The following schema illustrates the link between classes and tables to store objects in a relational database:



5.5.5. MAPPING OBJECTS TRONG OOM (2)

Defining data sources in an OOM

You create a data source to define where (in which database) data should be managed by OOM objects.

When you define a data source, you have to declare physical data models in the list of data source models. A data source can contain several models, you can select the source models among a list of models opened in the current workspace. All the selected models represent the same database that will store persistent objects from the OOM.

An OOM can contain several data sources.

Database connection

The Database Connection page lets you define the following parameters to connect to a database:

Parameter	Description
Using an ODBC data source	ODBC data source associated with the database
Using a connection string	Allows to connect to a database via other means than ODBC. A connection string is a series of semicolon delimited arguments that define parameters such as the data source provider and the location of the data source. For example, the connection string for jConnect is: `jdbc:sybase:Tds:machine-name:port-number`
User ID	User login
Password	User password

5.5.5. MAPPING OBJECTS TRONG OOM (3)

Defining O/R mapping

Object persistence requires to store and extract objects in a relational database. You can map OOM objects to PDM objects to create an object to relational mapping:

OOM object	Mapped to...
Class	Table
Attribute	Column
Operation	SELECT or UPDATE queries associated with a table
Association	Reference, view reference, table or view

There are different methods for creating an O/R mapping:

- Automatic mapping
- User-defined mapping

5.5.6. OPENING MỘT BIỂU ĐỒ ROSE TRONG OOM (4)

Opening a Rose model in an OOM

You can open a model built with Rational Rose (.MDL) version 98, 2000 and 2002.

A new OOM is created for the Rose model, and the objects of the Rose model are translated into OOM objects.

This functionality provides you with greater scope and flexibility. You can create an OOM from a Rose model, from which you generate Java files or objects for PowerBuilder to create applications. You can also use the OOM created from a Rose model to add to an existing OOM, or to generate a CDM or PDM for database analysis purposes.

The following table maps Rose languages to PowerDesigner languages:

Rose language	PowerDesigner language
CORBA	IDL-CORBA
Java	Java
C++	C++
VC++	C++
XML-DTD	XML-DTD
Visual Basic	Visual Basic 6
Analysis	Analysis
Oracle 8, Ada, COM, Web Modeler, and *all other languages*	Analysis

5.5.6. OPENING MỘT BIỂU ĐỒ ROSE TRONG OOM (5)

Importing general objects

When you open a Rose model in an OOM, the following objects are imported:

In a Rose model...	Becomes in an OOM
Package	Package
Diagram	Diagram
Note	Note
Note Link	Note Link
Text	Text
File	File

When you open a Rose model in an OOM, the following properties are imported:

In a Rose model...	Becomes in an OOM
Documentation	Comment
Zoom	Page scale
Stereotype	Stereotype

5.5.6. OPENING MỘT BIỂU ĐỒ ROSE TRONG OOM (6)

Importing use case diagrams

The following table maps Rose objects to OOM objects in the following way:

In a Rose model...	Becomes in an OOM
Class with <<actor>>, <<business actor>>, or <<business worker>> stereotype	Implementation class, and actor containing this implementation class (see paragraph below)
Use case	Use case
Generalization	Generalization
Association	Association
Dependency	Dependency

5.6. CÁC MÔ HÌNH ĐƯỢC SINH RA TỪ MH HƯỚNG ĐT

5.6.1. Giới thiệu tổng quan

5.6.2. Sinh tạo OOM từ OOM

5.6.3. Sinh tạo CDM từ OOM

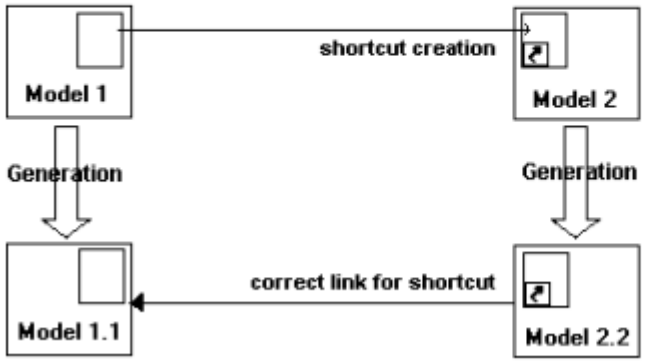
5.6.4. Sinh tạo PDM từ OOM

5.6.1. GIỚI THIỆU TỔNG QUAN

Generation basics

When you generate from an Object-Oriented Model you can generate to an OOM, a CDM, or a PDM. You can generate a model from a global OOM or from a package within the model. Limiting model generation to a single package is useful when different designers own packages of the same OOM. Designers can generate their packages independently from others. Generating a package results in an independent model.

The model generation process allows you to define the target object of a shortcut in a generated model.



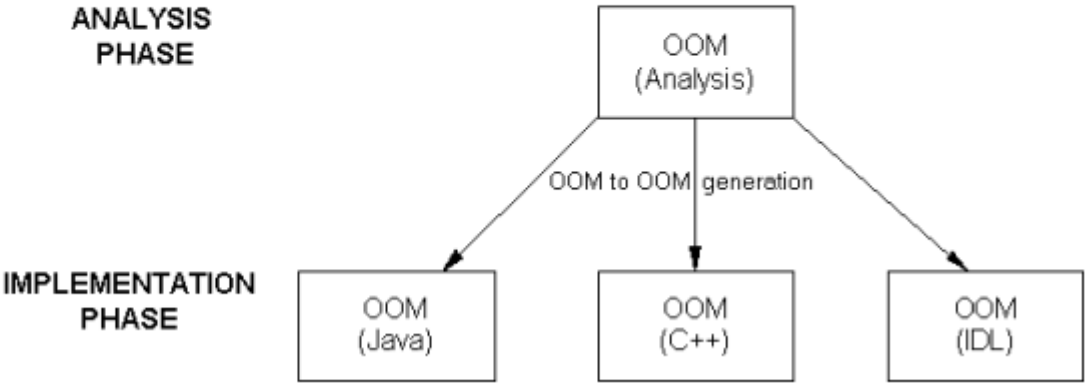
5.6.2. SINH TẠO OOM TỪ OOM

Why generating an OOM into an OOM?

You can generate an OOM into an OOM when you need to create versions of a model. This kind of generation allows you to create a copy of a given model and define generation links between objects in the source OOM and their equivalent in the generated OOM. When changes are made to the source model, they can then be easily propagated to the generated models using the Update Existing Model generation mode.

Implementing for different languages

You can use the OOM to OOM generation to generate an analytical OOM (designed with the Analysis object language) into implementation OOMs designed for the different object languages supported in PowerDesigner.



5.6.3. SINH TẠO CDM TỪ OOM

Generating CDM objects

If you have been working on a class diagram in an Object-Oriented Model (OOM) and want to change approach, you can generate a Conceptual Data Model (CDM) of an OOM class diagram. This process translates OOM objects into CDM objects and allows you to switch to an entity/relationships or Merise analysis.

You will then be able to further design your model and eventually generate a Physical Data Model (PDM) from the CDM.

When you generate a CDM from an OOM, PowerDesigner translates OOM objects and data types to CDM objects and data types.

The current object language of an
CDM. CDM generation translates

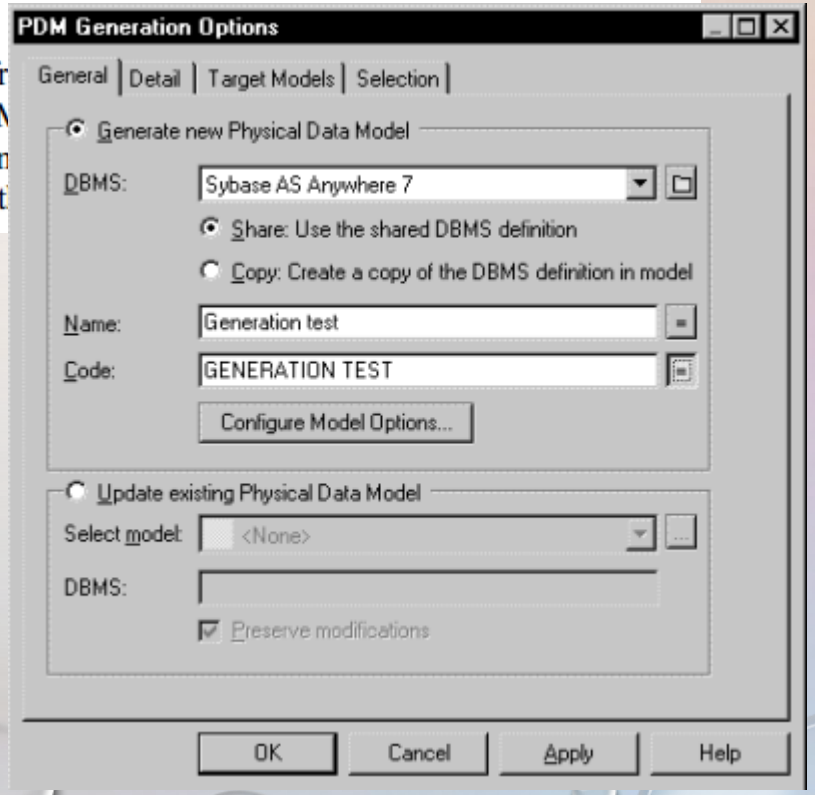
OOM object	CDM object after generation
Domain	Domain
Class	Entity
Interface	Not translated
Attribute	Attribute
Identifier	Identifier
Operation	Not translated
Association	Relationship or association
Association class	Entity/relationship notation: the association class becomes an entity with two associations. Merise notation: the class becomes an association, and the OOM associations will be generated as association links
Dependency	Not translated
Realization	Not translated
Generalization	Inheritance

5.6.4. SINH TẠO PDM TỪ OOM

Generating a Physical Data Model from an Object-Oriented Model

You may want to go from a class diagram of an Object-Oriented Model (OOM) to a Physical Data Model (PDM) because the design of your system in the OOM can be directly used by the database you want to work with. The advantage of this process is to model the objects in the world they live in and, through the generation, to automate the translation to tables and fields in the database.

When you generate a PDM fr
objects and data types to PDM
current Database Managemen
of an OOM has no effect on t



5.7. TỔNG KẾT CHƯƠNG & BÀI TẬP

Hãy vẽ mô hình OOM cho mô tả hệ thống. Sau đó, chuyển đổi sang các mô hình CDM và PDM.
(sẽ được cung cấp tại lớp học)

MÔN HỌC:

CÔNG CỤ THIẾT KẾ HỆ THỐNG THÔNG TIN

GIẢNG VIÊN: THS. VÕ THỊ KIM-ANH

(2021)