**In**

Cách vẽ Package Diagram

**Thực hiện bởi**

**Nguyễn Đăng Quang  
Vũ Thị Thành Vinh  
Cao Tiến Thắng  
Hoàng Văn Quyền  
Nguyễn Vũ Thanh Tùng**

**OOAD Nhóm 3**

**21 tháng 4 năm 2024**

**Mục lục**

[**0. Định nghĩa:**](#_heading=h.c5s3nph8igvk) **3**

[**1. Access**](#_heading=h.3znysh7) **3**

[**2. Dependency**](#_heading=h.sylew0ixx1i3) **4**

[**3. Import**](#_heading=h.aat40tg78ee1) **5**

[**4. Note**](#_heading=h.8hvl9qi9w28e) **6**

[**5. Realization**](#_heading=h.fiun9f1ixmdw) **7**

[**6. Constraint**](#_heading=h.cjp456u30va0) **8**

[**7. Generalization**](#_heading=h.pjahloo2h5n4) **9**

[**8. Merge**](#_heading=h.jbxitunnnbyw) **9**

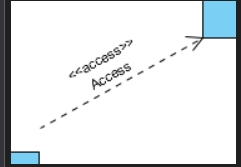
[**9. Package**](#_heading=h.7pjftw993irz) **10**

[**10. Subsystem**](#_heading=h.h12y9vxwgyce) **11**

# Định nghĩa:

Package diagram thể hiện sự sắp xếp và tổ chức các thành phần mô hình trong dự án có quy mô trung bình đến lớn. Package diagram có thể hiển thị cả cấu trúc sự phụ thuộc giữa các hệ thống con hoặc modules.

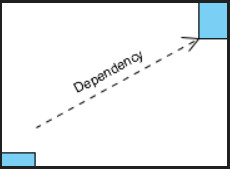
# [Access](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#access)

* 1. **Định nghĩa:** Một phần tử import được định nghĩa là một mối quan hệ có hướng giữa không gian tên import và phần tử có thể đóng gói. Tên của phần tử có thể đóng gói hoặc biệt danh (alias) của có được thêm vào không gian tên của không gian tên nhập. Cũng có thể kiểm soát liệu phần tử nhập có thể được nhập tiếp hay không.  
     

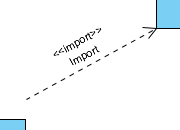
Một phần tử nhập được hiển thị bằng một mũi tên nét đứt và đầu mũi tên mở từ không gian tên nhập đến phần tử nhập. Từ khóa <<import>> được hiển thị gần mũi tên nét đứt nếu visibility là public, ngược lại <<access>> thể hiện visibility là private.

* 1. **Thuộc tính:**
     1. **Tên:** Tên của quan hệ access
     2. **Supplier:** **Phần tử độc lập với phần tử của client, trong cùng một ngữ cảnh và cùng mối quan hệ phụ thuộc**. Trong mối quan hệ phụ thuộc có hướng (Refinement Abstractions), một số quy ước thông thường trong lĩnh vực phần mềm hướng đối tượng lớp đặt các phần tử trừu tượng vào vai trò này. Mặc dù có quy ước này, người dùng UML có thể quy định sự phụ thuộc phù hợp hơn với lĩnh vực của họ, làm cho phần tử trừu tượng phụ thuộc vào một phần tử cụ thể hơn.
     3. **Client:** **Phần tử phục thuộc vào phần tử cung cấp.** Trong một số trường hợp (như Trace Abstraction), việc gán định hướng (nghĩa là, việc chỉ định phần tử client) là quyền tự quyết của người làm mô hình hóa, và là một quy ước.
     4. **Visibility:** Xác định visibility của PackageableElement (phần tử đóng gói) được nhập vào importing Package. Mặc định visibility giống với visibility của phần tử nhập. Nêu phần tử nhập không có visibility, có thể thêm vào phần tử nhập. Mặc định là public.
     5. **Tài liệu:** Mô tả về quan hệ truy cập.

# [Dependency](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#dependency)

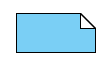
* 1. **Định nghĩa:** Phụ thuộc là một mối quan hệ thể hiện rằng một hoặc một tập hợp các phần tử mô hình đòi hỏi phải có phần tử mô hình khác để có thể định nghĩa hoặc triển khai chúng. Điều này có nghĩa là toàn bộ ngữ nghĩa của phần tử phụ thuộc phụ thuộc vào ngữ nghĩa hoặc cấu trúc của định nghĩa của phần tử cung cấp (Supplier element(s)).
  2. **Thuộc tính:**
     1. **Tên:** Tên của phụ thuộc
     2. **Supplier:** **Phần tử độc lập với phần tử của client, trong cùng một ngữ cảnh và cùng mối quan hệ phụ thuộc**. Trong mối quan hệ phụ thuộc có hướng (Refinement Abstractions), một số quy ước thông thường trong lĩnh vực phần mềm hướng đối tượng lớp đặt các phần tử trừu tượng vào vai trò này. Mặc dù có quy ước này, người dùng UML có thể quy định sự phụ thuộc phù hợp hơn với lĩnh vực của họ, làm cho phần tử trừu tượng phụ thuộc vào một phần tử cụ thể hơn.
     3. **Client:** **Phần tử phục thuộc vào phần tử cung cấp.** Trong một số trường hợp (như Trace Abstraction), việc gán định hướng (nghĩa là, việc chỉ định phần tử client) là quyền tự quyết của người làm mô hình hóa, và là một quy ước.
     4. **Visibility:** Xác định nơi mà sự phụ thuộc xuất hiện trong các không gian tên khác nhau trong mô hình tổng thể và khả năng truy cập của nó.
     5. **Tài liệu:** Mô tả về phụ thuộc.

# [Import](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#import)

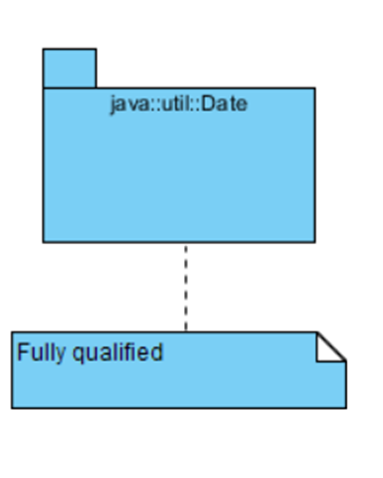
****

* 1. **Định nghĩa**: Một package được import được định nghĩa là mối quan hệ trực tiếp xác định package mà những thành phần trong package đấy sẽ được import bởi một namespace khác.
  2. **Thuộc tính**
     1. **Name**: Tên của mối quan hệ import
     2. **Supplier**: Các element độc lập với các client element, trong cùng mối quan hệ phụ thuộc. Trong một số mối quan hệ phụ thuộc được định hướng (như Refinement Abstractions), một quy ước chung trong miền phần mềm hướng đối tượng dựa trên lớp (Class-based object-oriented software) là đặt phần tử trừu tượng hơn vào vai trò “Supplier”. Người dùng UML có thể quy định về sự phụ thuộc phù hợp với domain của người dùng. Điều này làm cho một phần tử trừu tượng hơn phụ thuộc vào phần tử cụ thể hơn
     3. **Client**: Các element phụ thuộc vào các element là “supplier”. Trong một số trường hợp (như Trace Abstraction), việc định hướng (xác định element đóng vai trò “client”) sẽ phụ thuộc vào quyết định của người viết mô hình
     4. **Visibility**: Chỉ định mức độ hiển thị của các element có thể được đóng gói **(PackageableElements)** đã được import trong namespace, tức là xác định xem liệu các element được import, tức là có được hiển thị với các package khác sử dụng **importingPackage** với vai trò là một **importedPackage**. Nếu như package được các element import đến ở dạng public, các element sẽ được hiển thị bên ngoài package. Ngược lại, nếu package ở dạng private thì các element sẽ không được hiển thị
     5. **Description**: Mô tả của mối quan hệ import

# [Note](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#note)

****

* 1. **Định nghĩa:** Note (hay comment) cung cấp khả năng đính kèm các nhận xét khác nhau vào các element. Nhận xét này không ảnh hưởng đến ý nghĩa của các element, nhưng chứa thông tin hữu ích cho người viết mô hình.
  2. **Thuộc tính**
     1. **Name:** Tên của “Note”
     2. **Documentation:** một chuỗi kí tự đóng vai trò là comment
  3. **Ví dụ**

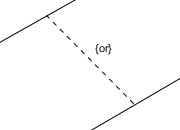


* Note có nội dung “Fully qualified” được sử dụng để chứa thông tin cho package “java::util::Date” biểu thị cách viết tên đầy đủ của một package

# [Realization](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#realization)

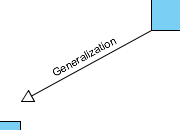
* 1. **Định nghĩa:** Realization là một mối quan hệ trừu tượng hóa chuyên biệt giữa 2 tập hợp các phần tử mô hình (model element), một tập hợp đại diện cho một đặc tả kỹ thuật (phía supplier), một đại diện cho việc triển khai về sau (phía client). Realization có thể được sử dụng để mô hình hóa sàng lọc từng bước, tối ưu hóa, chuyển đổi, tổng hợp mô hình, …
  2. **Thuộc tính**
     1. **Name:** Tên của mối quan hệ hiện thực hóa
     2. **Supplier:** Các phần tử (element) độc lập với các phần tử phía client, trong cùng mối quan hệ phụ thuộc. Trong một số mối quan hệ phụ thuộc được định hướng (như Refinement Abstractions), một quy ước chung trong lĩnh vực phần mềm hướng đối tượng dựa trên lớp, là đặt phần tử trừu tượng hơn vào vai trò “Supplier”. Bất chấp quy ước trên, người dùng UML có thể quy định ý nghĩa của sự phụ thuộc phù hợp với lĩnh vực mà họ muốn. Điều này làm cho một phần tử trừu tượng hơn phụ thuộc vào phần tử cụ thể hơn
     3. **Client:** Các phần tử phụ thuộc vào các phần tử là “Supplier”. Trong một số trường hợp (như Trace Abstraction), việc định hướng (chỉ định phần tử là “client”) sẽ phụ thuộc vào quyết định của người lập mô hình UML
     4. **Visibility:** Xác định vị trí mối quan hệ hiện thực hóa “relization” xuất hiện trong các namespace khác nhau trong mô hình tổng thể và khả năng truy cập của nó
     5. **Mapping:** Là thành phần của một “Expression”, nêu rõ mối quan hệ trừu tượng giữa “supplier” và “client”. Trong một số trường hợp, chẳng hạn như “Derivation”, mapping thường mang tính hình thức và một chiều. Trong trường hợp khác, chẳng hạn như “Trace”, mapping thường mang tính chất không chính thức và hai chiều. Biểu thực ánh xạ (mapping expression) là tùy chọn và có thể bị bỏ qua nếu mối quan hệ chính xác giữa các phần tử không được chỉ định
     6. **Documentation:** Mô tả mối quan hệ hiện thực hóa.

# [Constraint](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#constraint)

****

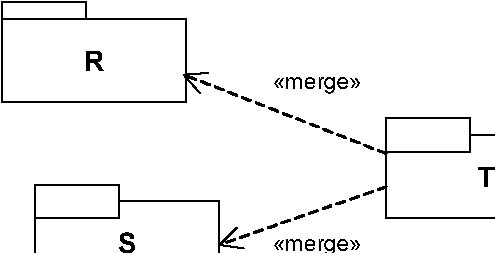
* 1. **Định nghĩa:** Một điều kiện hoặc hạn chế được thể hiện bằng văn bản ngôn ngữ tự nhiên hoặc ngôn ngữ máy có thể đọc được nhằm mục đích khai báo một số ngữ nghĩa của một phần tử
  2. **Thuộc tính**
     1. **Name:** Tên của ràng buộc (constraint), là tùy chọn và thường được bỏ qua
     2. **Expression:** Điều kiện cần phải đúng khi được đánh giá để thỏa mãn ràng buộc
     3. **Documentation:** Mô tả của ràng buộc

# [Generalization](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#generalization)

****

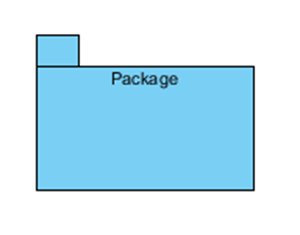
* 1. **Định nghĩa:** Khái quát hóa là mối quan hệ giữa các lớp trong đó một lớp chia sẻ cấu trúc và/hoặc hành vi với một hoặc nhiều lớp khác, là mối quan hệ giữa một lớp tổng quát hơn và một lớp cụ thể hơn. Mỗi thực thể của lớp cụ thể hơn là một thực thể gián tiếp của lớp khái quát. Do đó, các lớp cụ thể hơn kế thừa các đặc điểm các lớp khái quát hơn.
  2. **Thuộc tính**
     1. **Tên:** Tên của mối quan hệ khái quát hóa
     2. **Khái quát:** Tham chiếu đến lớp khái quát trong quan hệ khái quát hóa.
     3. **Cụ thể:** Tham chiếu đến lớp cụ thể trong mối quan hệ khái quát hóa.
     4. **Visibility:** Xác định nơi mối quan hệ khái quát hóa xuất hiện trong các namespace khác nhau trong mô hình tổng thể và khả năng truy cập của mối quan hệ này.
     5. **Tài liệu:** Mô tả về mối quan hệ khái quát hóa
     6. **Tính thay thế:** Chỉ ra rằng các lớp cụ thể hơn có thể được sử dụng nơi mà lớp khái quát hơn được sử dụng. Nếu đúng, các dấu vết thực thi của lớp cụ thể hơn sẽ là tập con của các dấu vết thực thi của lớp khái quát hơn.

# [Merge](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#merge)

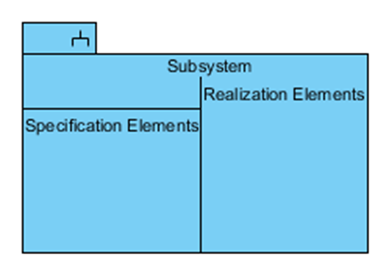
****

* 1. **Định nghĩa:** Hợp nhất là một mối quan hệ có hướng giữa hai gói cho biết rằng nội dung của hai gói sẽ được kết hợp. Nó rất giống với Khái quát hóa trong ý nghĩa là phần tử nguồn sẽ thêm các đặc điểm của phần tử mục tiêu vào các đặc tính của chính nó, dẫn đến một phần tử kết hợp các đặc tính của cả hai.  
     Cơ chế này nên được sử dụng khi các phần tử được định nghĩa trong các gói khác nhau có cùng tên và được dự định đại diện cho cùng một khái niệm. Thường thì nó được sử dụng để cung cấp các định nghĩa khác nhau của một khái niệm nhất định cho các mục đích khác nhau, bắt đầu từ một định nghĩa cơ bản chung. Một khái niệm cơ bản nhất định được mở rộng theo từng bước, với mỗi bước được định nghĩa trong một gói hợp nhất riêng biệt. Bằng cách chọn các bước nào được hợp nhất, có thể thu được một định nghĩa tùy chỉnh của một khái niệm cho một mục đích cuối cùng cụ thể. Gói hợp nhất đặc biệt hữu ích trong meta-modeling và được sử dụng rộng rãi trong việc định nghĩa của meta-model UML.  
     Một cách khái niệm, một gói hợp nhất có thể được xem như một hoạt động lấy nội dung của hai gói và tạo ra một gói mới kết hợp nội dung của các gói liên quan trong quá trình hợp nhất. Về mặt ý nghĩa của mô hình, không có sự khác biệt giữa một mô hình có các gói hợp nhất rõ ràng và một mô hình trong đó tất cả các hợp nhất đã được thực hiện.
  2. **Thuộc tính**
     1. **Tên:** Tên của mối quan hệ hợp nhất
     2. **Supplier:** Các phần tử độc lập với các phần tử client, trong cùng một mối quan hệ phụ thuộc. Trong một số mối quan hệ phụ thuộc có hướng (như Refined Abstractions), một quy ước phổ biến trong lĩnh vực phần mềm hướng đối tượng là đặt phần tử trừu tượng hơn vào vai trò này. Mặc dù có quy ước này, người vẽ UML có thể quy định một ý nghĩa của sự phụ thuộc phù hợp với lĩnh vực của họ, làm cho một phần tử trừu tượng hơn phụ thuộc vào phần tử cụ thể hơn.
     3. **Client:** Các phần tử phụ thuộc vào các phần tử Supplier. Trong một số trường hợp (ví dụ như Trace Abstraction) việc định hướng (tức là, việc chỉ định phần tử Client) là do quy ước của người mô hình hóa chúng.
     4. **Visibility:** Xác định nơi mối quan hệ hợp nhất xuất hiện trong các namespace khác nhau trong mô hình tổng thể và khả năng truy cập của mối quan hệ này.
     5. **Tài liệu:** Tài liệu của mối quan hệ hợp nhất.

# [Package](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#package)

* 1. **Định nghĩa:** Một gói được sử dụng để nhóm các phần tử và cung cấp một không gian tên (namespace) cho những phần tử đó. Một gói là không gian tên (namespace) cho chính những thành viên của nó, một gói có thể chứa nhiều gói khác.
  2. **Thuộc tính**
     1. **Tên:** Tên của gói
     2. **Parent (cha):** Phần tử chứa gói được đề cập.
     3. **Visibility (Khả năng truy cập):** Xác định vị trí của mối quan hệ gộp (merge relationship) xuất hiện trong các không gian tên (namespace) khác nhau trong mô hình tổng thể, và sự truy cập vào gói đó. Xác định cách mà gói và các thành phần của nó có thể được truy cập và sử dụng từ các phạm vi khác trong hệ thống.
     4. **Documentation (tài liệu):** Mô tả của gói.
     5. **Abstract (trừu tượng):** Nếu đúng, gói không cung cấp một khai báo hoàn chỉnh và thường không thể được khởi tạo. Một gói trừu tượng được sử dụng bởi các gói khác.
     6. **Leaf (lá):** Cho biết liệu có thể tiếp tục chuyên biệt hóa một gói hay không. Nếu giá trị là “đúng” thì không thể tiếp tục chuyên biệt hóa gói đó.
     7. **Root (gốc):** Cho biết liệu gói có tổ tiên (ancestor) hay không. (nhận giá trị đúng khi gói không có tổ tiên)
     8. **Children (các con):** Các con của gói

# [Subsystem](https://www.visual-paradigm.com/VPGallery/diagrams/Package.html#subsystem)

* 1. **Định nghĩa:** Một hệ thống con biểu diễn ranh giới của hệ thống con vật lý.
  2. **Tính chất**
     1. **Name (tên):** Tên của hệ thống con.
     2. **Parent (cha):** Phần tử chứa hệ thống con được đề cập.
     3. **Abstract (trừu tượng):** Nếu đúng, hệ thống con không cung cấp một khai báo hoàn chỉnh và thường không thể được khởi tạo. Một hệ thống con trừu tượng được sử dụng bởi các hệ thống con khác.
     4. **Leaf (lá):** Cho biết liệu có thể tiếp tục chuyên biệt hóa hệ thống con hay không. Nếu giá trị là “đúng” thì không thể tiếp tục chuyên biệt hóa hệ thống con được đề cập.
     5. **Root (gốc):** Cho biết hệ thống con có tổ tiên (ancestor) hay không. (nhận giá trị đúng khi hệ thống con không có tổ tiên)
     6. **Operations (các thao tác):** Một thao tác là một tính năng hành vi của hệ thống con, chỉ định tên, loại, các tham số và ràng buộc cho việc gọi, thực hiện hành vi tương ứng. Các thao tác ở đây đề cập đến các thao tác được sở hữu bởi hệ thống con.
     7. **Children (các con):** Các con của hệ thống con.