Package Diagram

Outline:

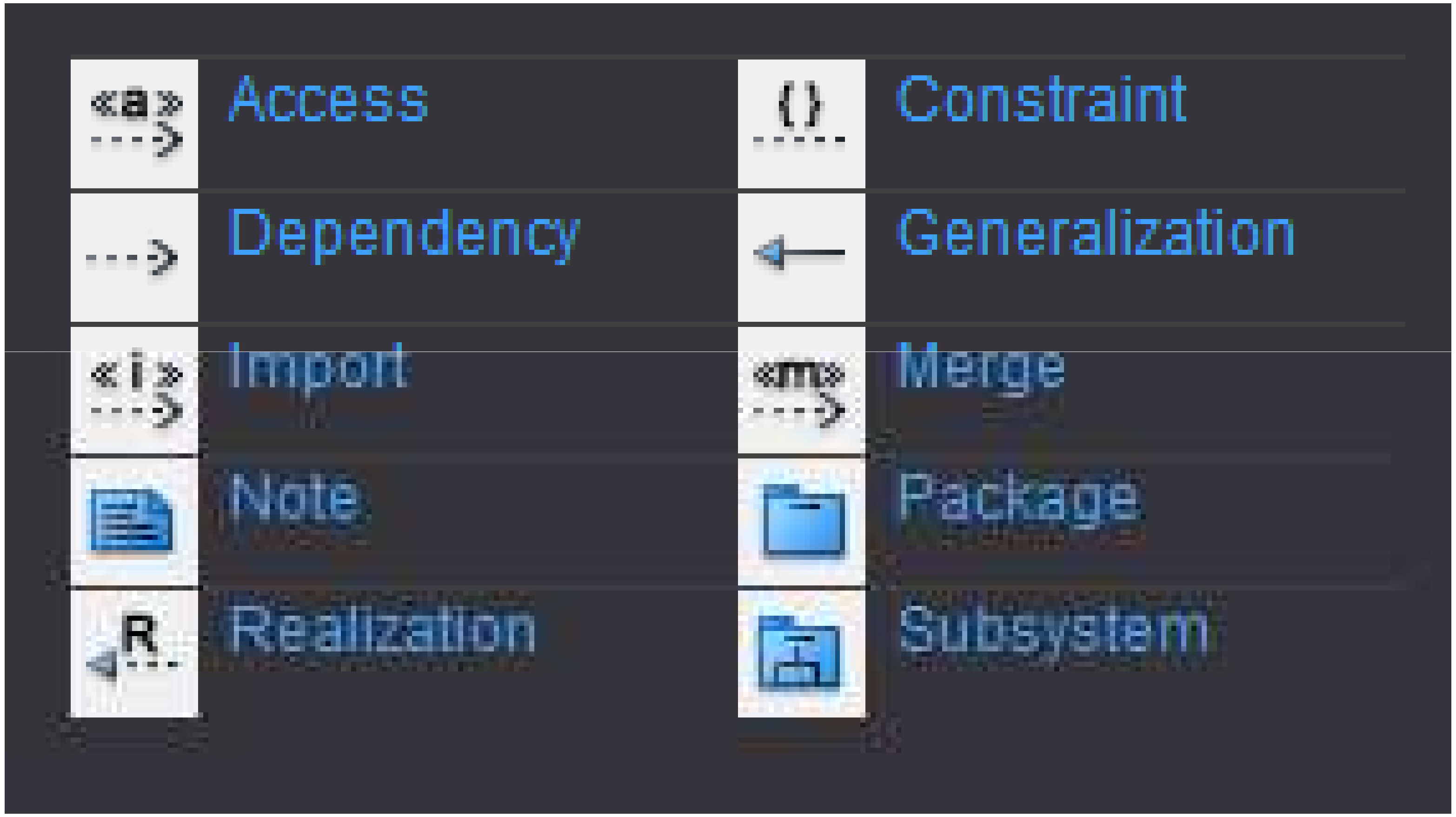
- Introduction.
- Package.
- What is Packageable Element?
- Relationship between Packages.
- Element Import.
- Package Import.
- Package Merge.
- Package Model.
- Use-case package Diagram.
- Class Package Diagram.
- References.
- Question ?????????

Package Diagram:

Package Diagram:

- A **package diagram** in the Unified Modeling Language depicts the dependencies between the packages that make up a model.
- Package diagram shows the arrangement and organization of model elements in middle to large scale project.
- Package diagram can also show both structure and dependencies between sub-systems or modules.

UML Package Symbols:



UML Package Symbols:

- Access: An element import is defined as a directed relationship between an importing namespace and a packageable element.
- Dependency: A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
- Import:A package import is defined as a directed relationship that identifies a package whose members are to be imported by a namespace.
- Merge: A package merge is a directed relationship between two packages that indicates that the contents of the two packages are to be combined.

Package:

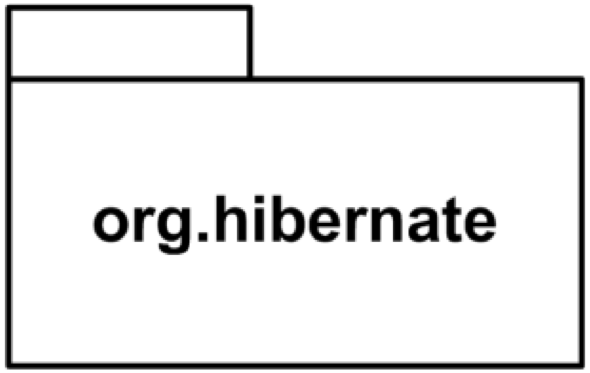
- A **package** is used to group elements, and provides a namespace for the grouped elements. A package is a namespace for its members, and may contain other packages.
- **Owned members** of a package should all be package elements.
- Package can also be merge with other package, thus provide the hierarchical organization of the package.
- Different types of elements are allow to have the same name.
- The members of the package may be shown within the boundaries of the package.

Package:

- The elements that can be referred to within a package using **non-qualified** names are: Owned Element, Imported Element, and elements enclosing namespaces.
- Owned and imported elements may have a **visibility** that determines whether they are available outside the package.

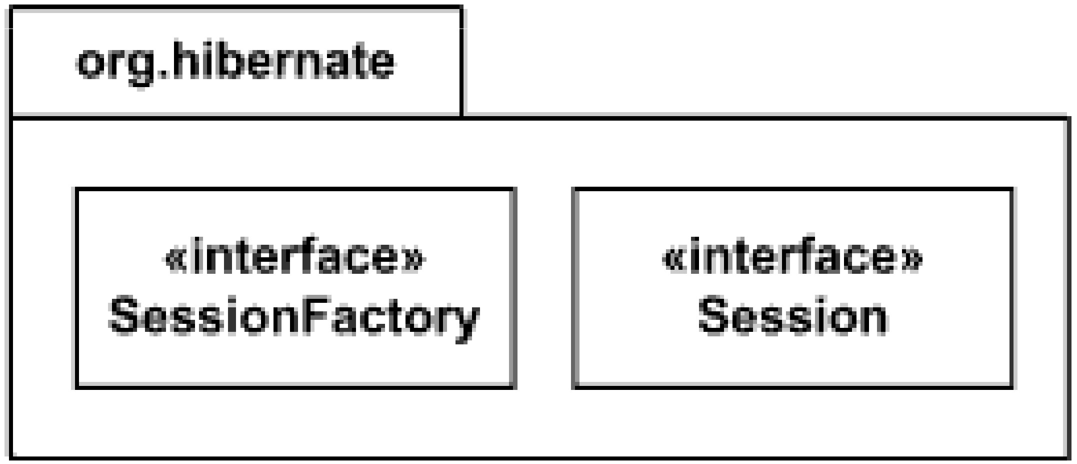
Package:

- Package Member are not shown inside the package.
- Package org.hibernate



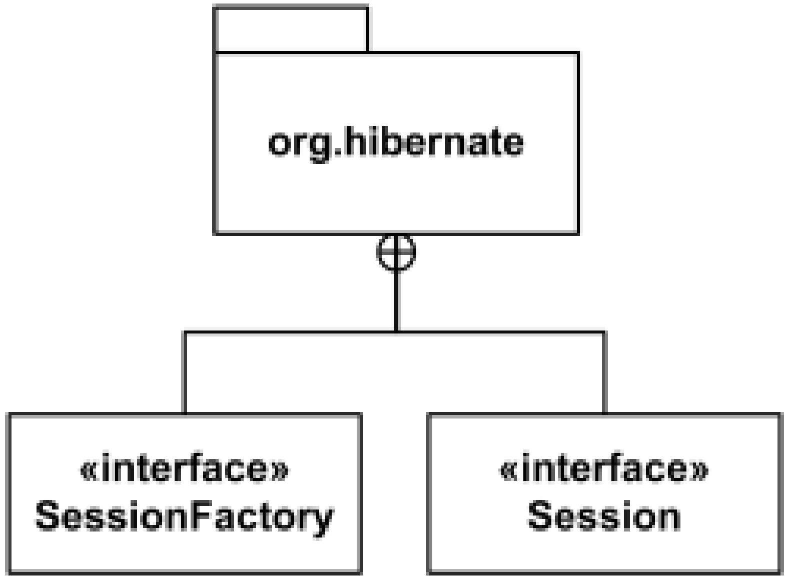
Package:

- Package org.hibernate contains SessionFactory and Session.
- Package Member are shown inside the package.



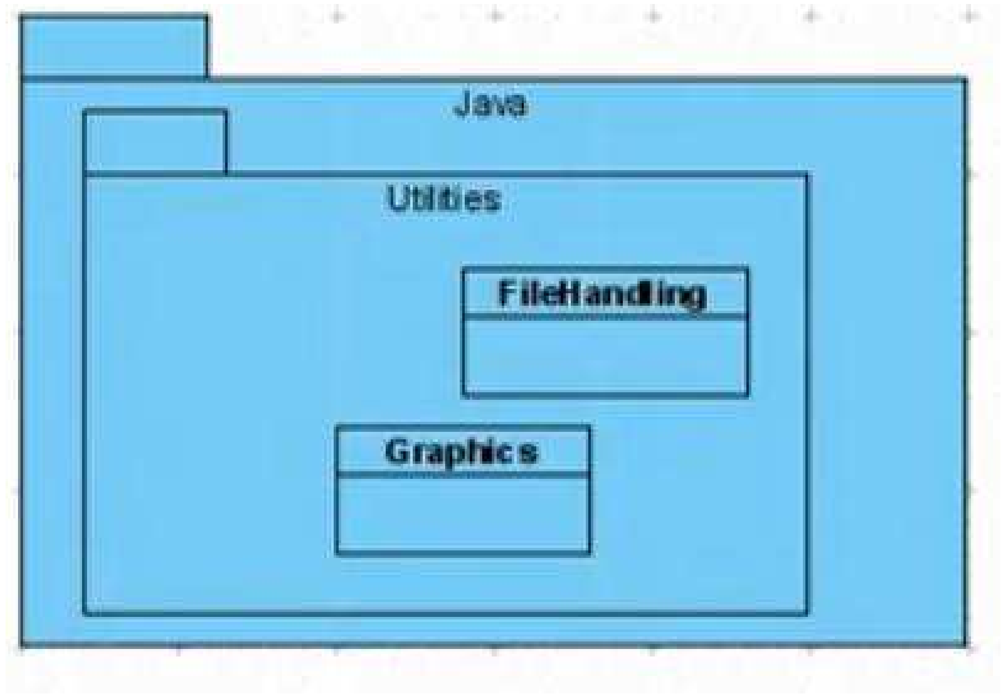
Package:

- Members of the package may be shown **outside** of the package by branching lines.
- Package org.hibernate contains interfaces Session and SessionFactory.



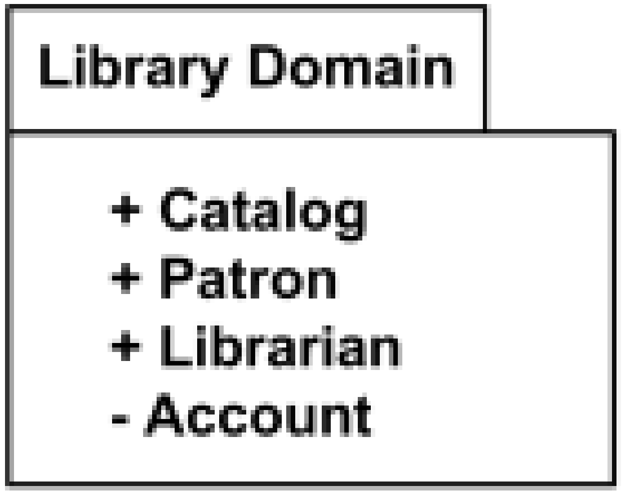
Package:

- Packages are useful for simplify this kind of diagrams
- Nested packages.
- Qualifier for Graphics class is Java::Utilities::Graphics



Package:

- **Visibility** of Owned and Import element.
- "+" for public and "-" for private or helper class.
- All elements of Library Domain package are public except for Account.



Packageable Element:

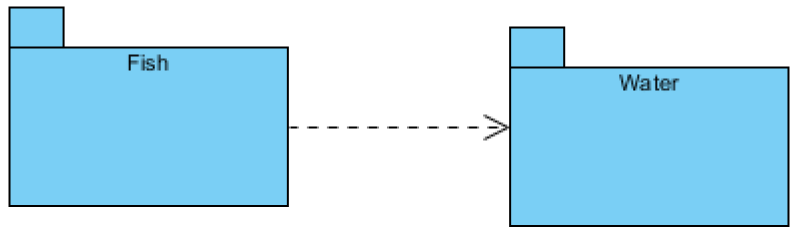
- **Packageable element** is a named element that may be **owned** directly by a package.
- Owned member of the package should all be **packageable elements**.
- If a package is removed from the model, so are all the elements owned by the package. Package by itself is **packageable element**, so any package could be also a member of the other packages.

Relationships:

- Dependency
- Generalization
- Refinement

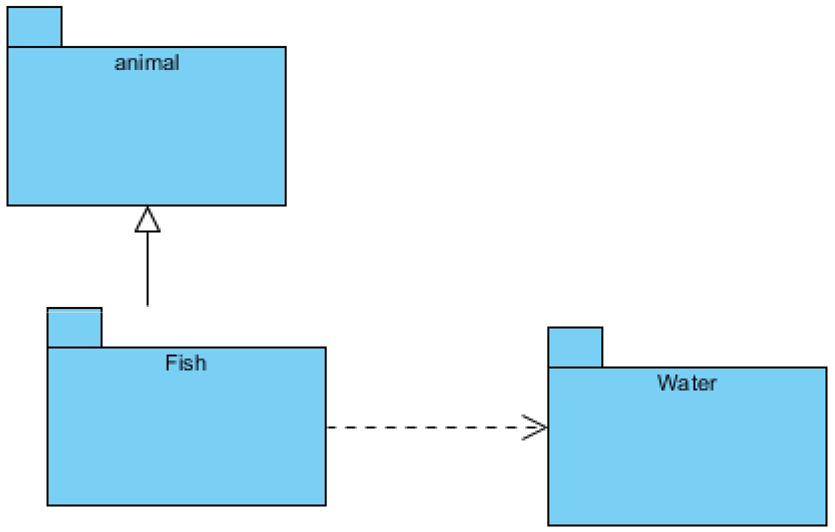
Dependency:

- One Package depends on another package.



Fish depends on water.

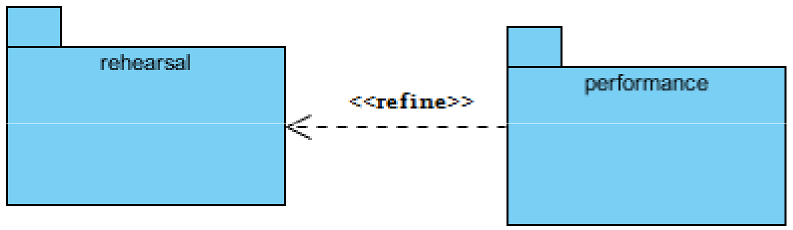
Generalizations:



Fish is a kind of Animal.

Refinement:

- Refinement shows different kind of relationship between packages.
- One Package refines another package, if it contains same elements but offers more details about those elements.

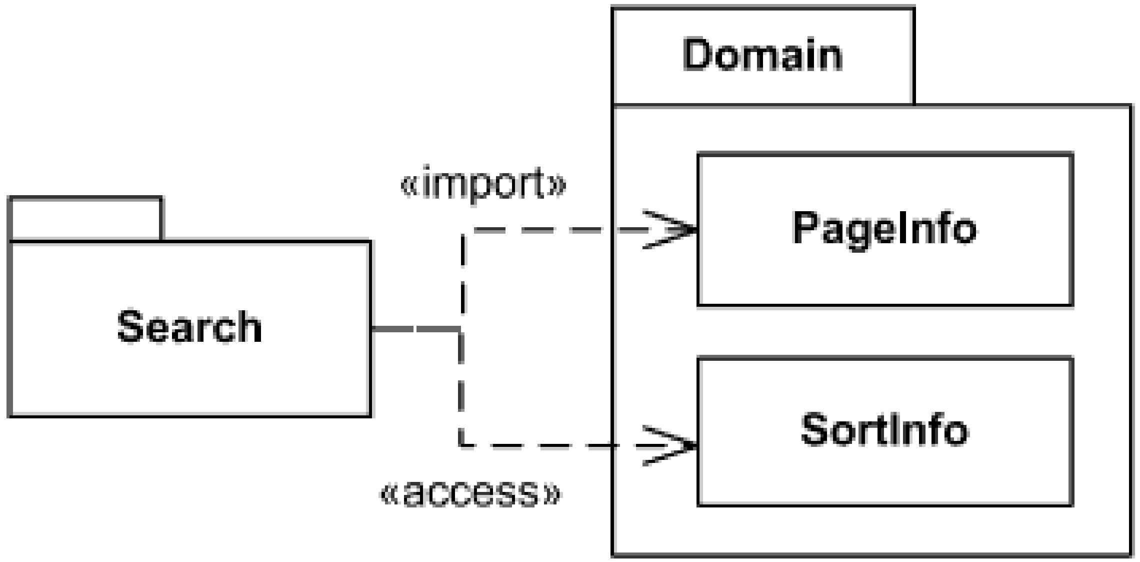


Performance refines rehersals.

Nodes and Edges:

Element Import:

- The keyword **«import»** is shown near the dashed arrow if the visibility is **public**
- The keyword **«access»** is shown to indicate **private** visibility
- Public import of PageInfo element and private import of SortInfo element from Domain package.

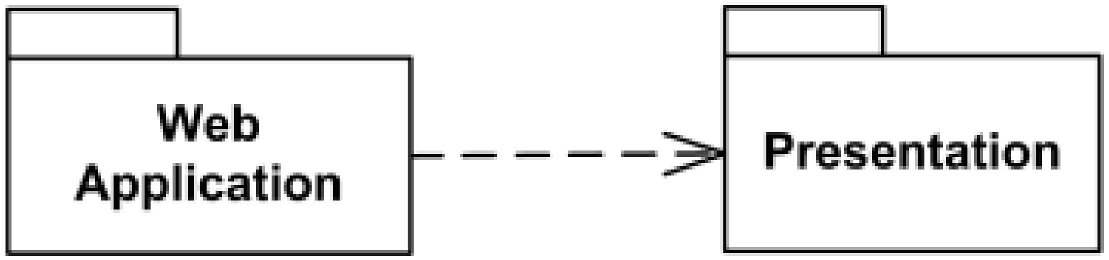


Package Import:

- **Package Import (PackageImport)** is a directed relationship between an importing **namespace** and imported **package**
- A package import is shown using a dashed arrow with an open arrowhead from the importing namespace to the imported package.
- It looks exactly like dependency and usage relationships.
- The **visibility** of a PackageImport could be either public or private.

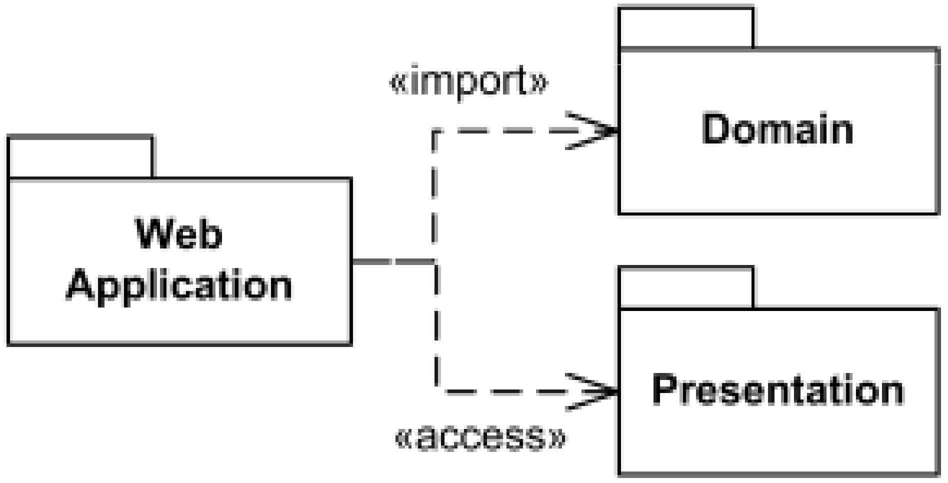
Package Import:

- The keyword **«access»** is shown to indicate **private** visibility.
- Public import of PageInfo element and private import of SortInfo element from Domain package.
- Web Application imports Presentation package.



Package Import :

- **Private import** of Presentation package and **public import** of Domain package.



Package Merge:

- A package merge is a directed relationship between two packages.
- It indicates that content of one package is extended by the contents of another package.
- Package merge used when elements defined in different packages have the same name and are intended to represent the same concept.
- Package merge is shown using a dashed line with an open arrowhead pointing from the receiving package to the merged package.

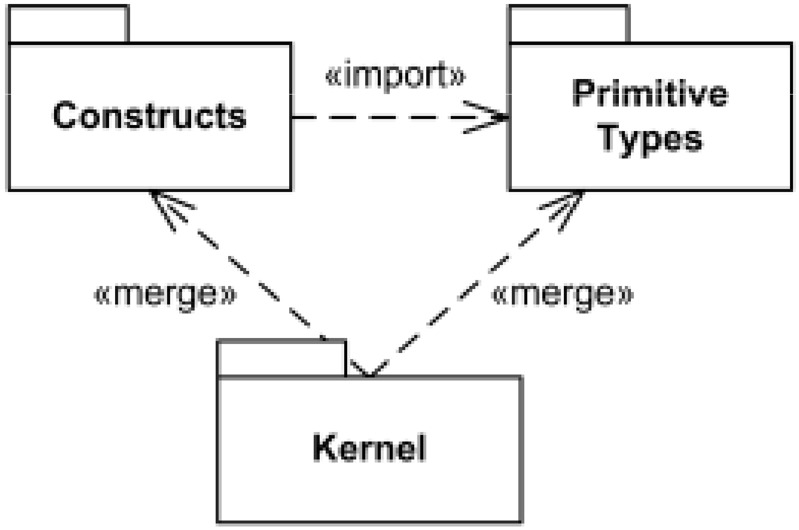
Package Merge :

Rules for Merging Packages:

- Private elements within the package do not merge with the receiving package.
- UML allows multiple inheritance in package merge.
- Any sub packages within the package are added to the receiving package.
- If both packages have different packages of the same name, a merge takes place between those packages.

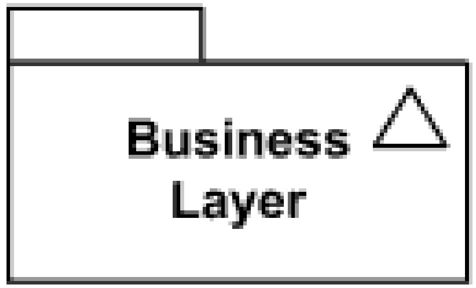
Package Merge :

- UML packages Constructs and Primitive Types are **merged** by UML Kernel package.



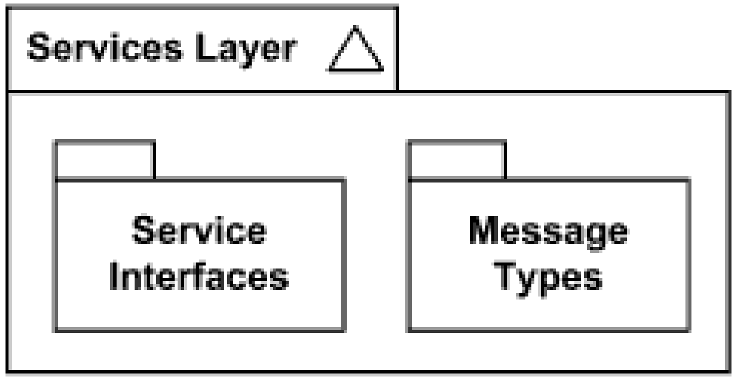
Model:

- **Model** is a package which captures a view of a system.
- View of the system defined by its purpose and abstraction level.
- Model is notated using the ordinary package symbol (a folder icon) with a small triangle in the upper right corner of the large rectangle.
- Business layer model :



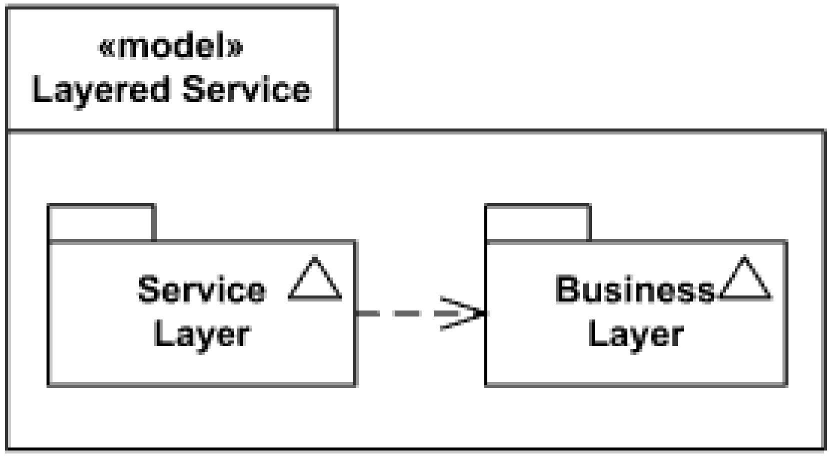
Model :

- If contents of the model are shown within the large rectangle, the triangle may be drawn to the right of the model name in the tab.
- Service Layer model contains service interfaces and message types.



Model:

- Model could be notated as a package with the keyword «model» placed above the name of the model.
- Stereotyped model Layered Service :

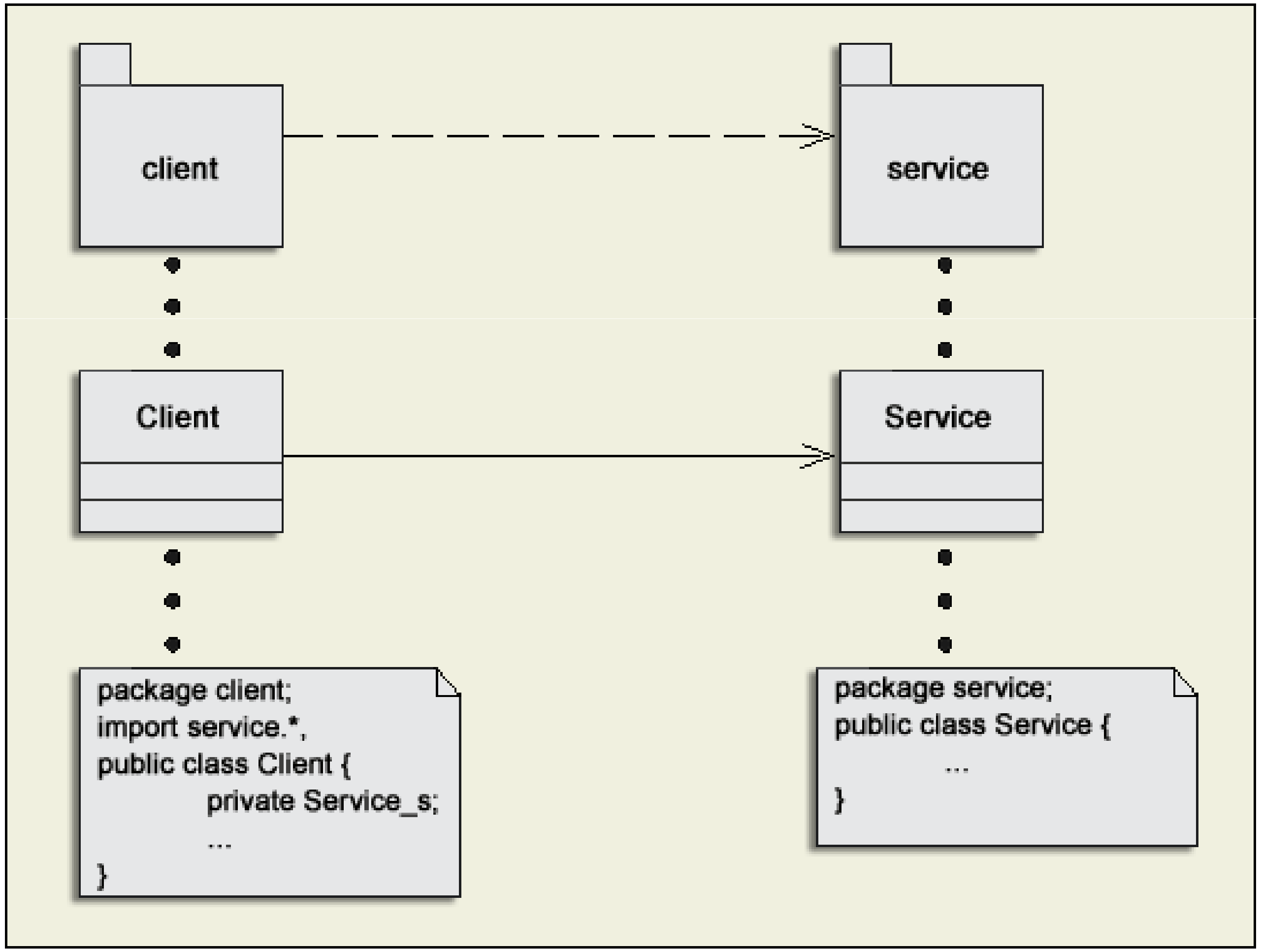


Package Relationship:

- A relationship between two packages is called a package dependency.
- Dependencies are not transitive.
- The dependency relationship between packages is consistent with the associative relationship between classes.
- Ex. If changing the contents of a package, P2, affects the contents of another package, P1, we can say that P1 has a Package Dependency on P2.

Package Relationship:

- Package dependency diagram :

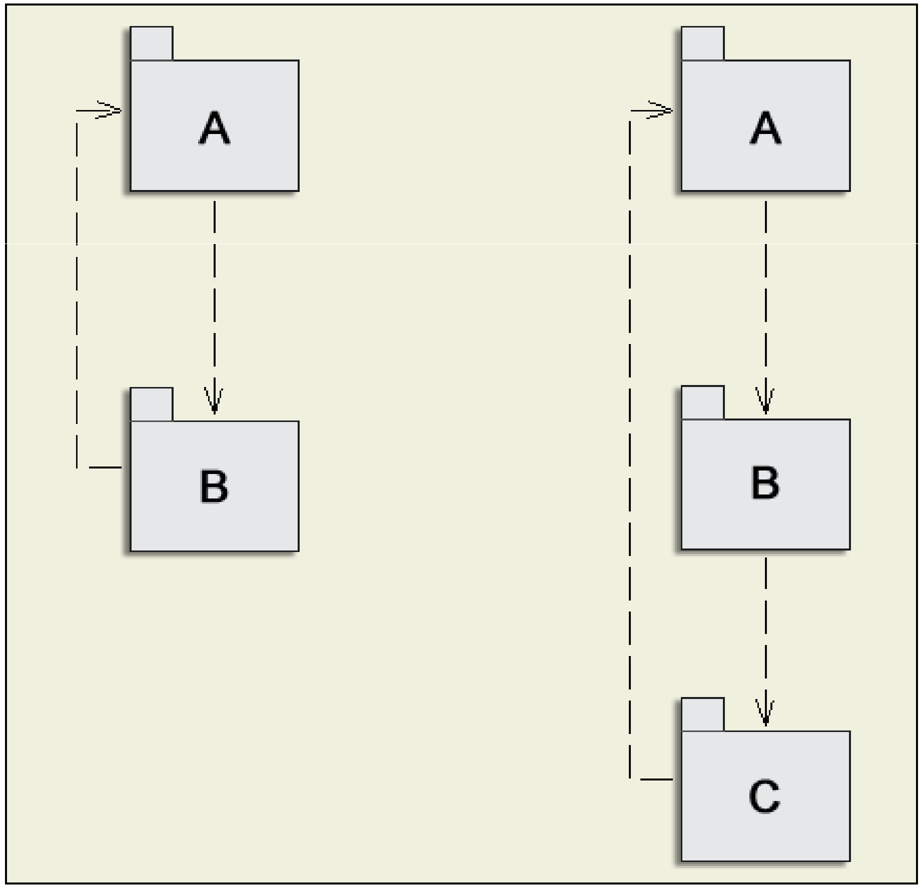


```
package client;
import service.*,
public class Client {
        private Service_s;
        ...
}
```

```
package service;
public class Service {
        ...
}
```

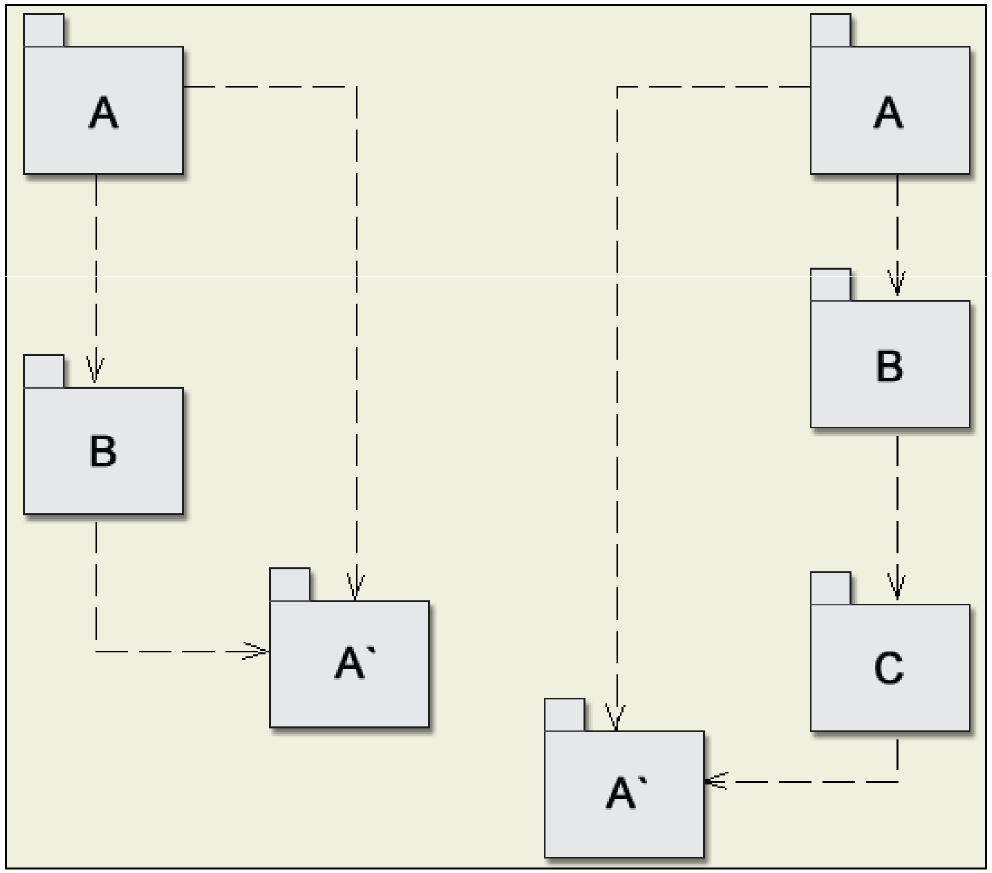
Package Relationship:

- Two types of relationship: Unidirectional and Bidirectional
- Unidirectional Diagram :



Package Relationship:

- Bidirectional Diagram :

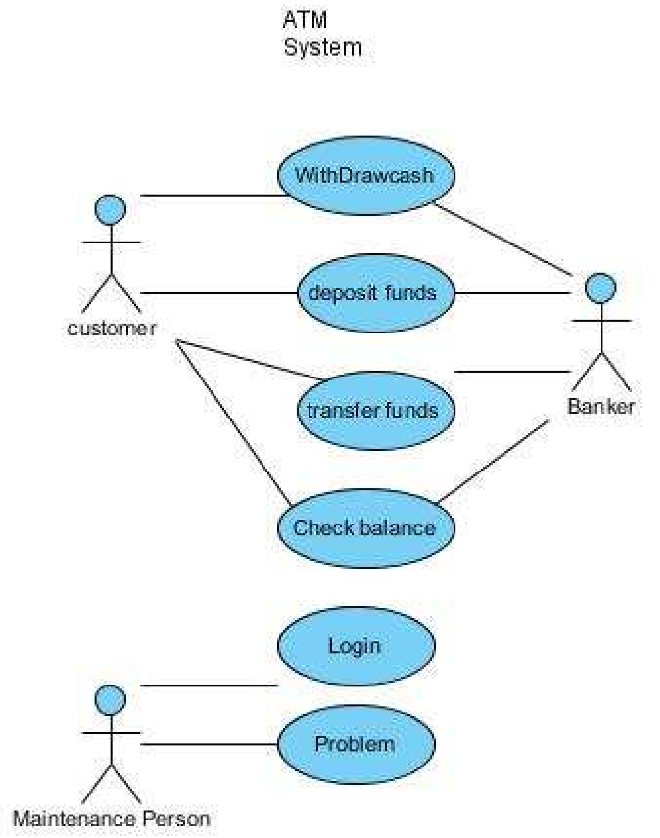


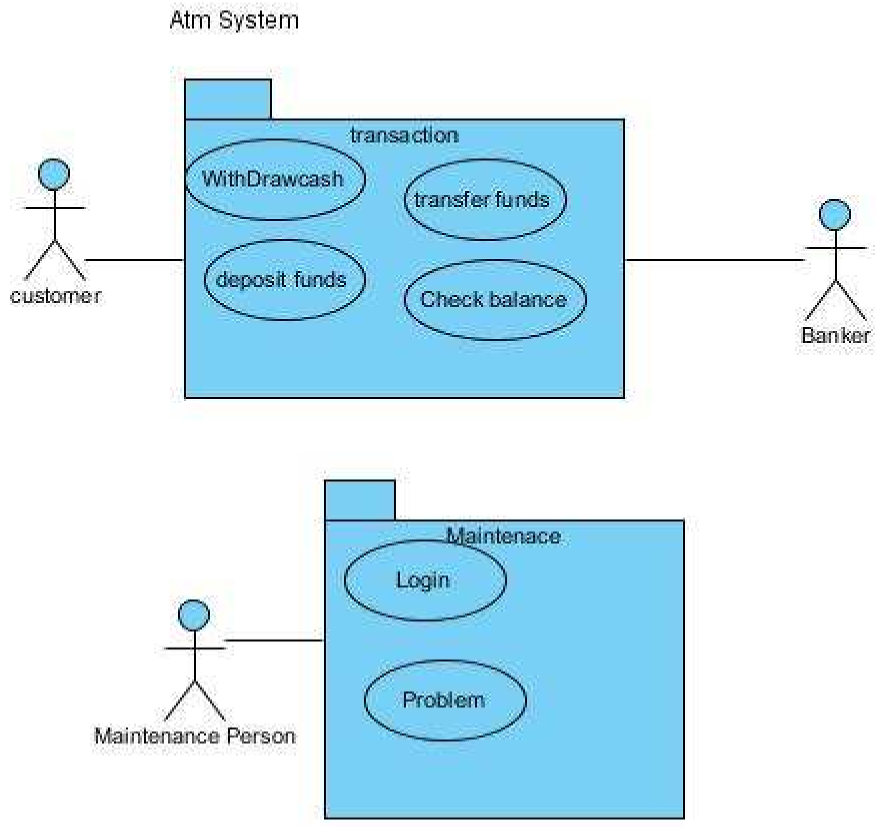
Package Relationship:

- Package Coupling—Unidirectional relationships between packages emphasize lower coupling. While some coupling must exist, those packages most loosely coupled are more easily maintained.
- Reuse Impact—Packages with bidirectional dependencies limit reusability. Those packages exhibiting lower degrees of coupling on other packages promote reuse.
- Layering—Defining unidirectional dependencies is consistent with how we would layer a system. Typically, upper-level layers are dependent on lower-level layers. A package should reside in, not span across, a layer. As such, packages in lower-level layers should be less dependent on other packages, increasing the reusability of those packages.

Examples:

Use Case Package Diagram:

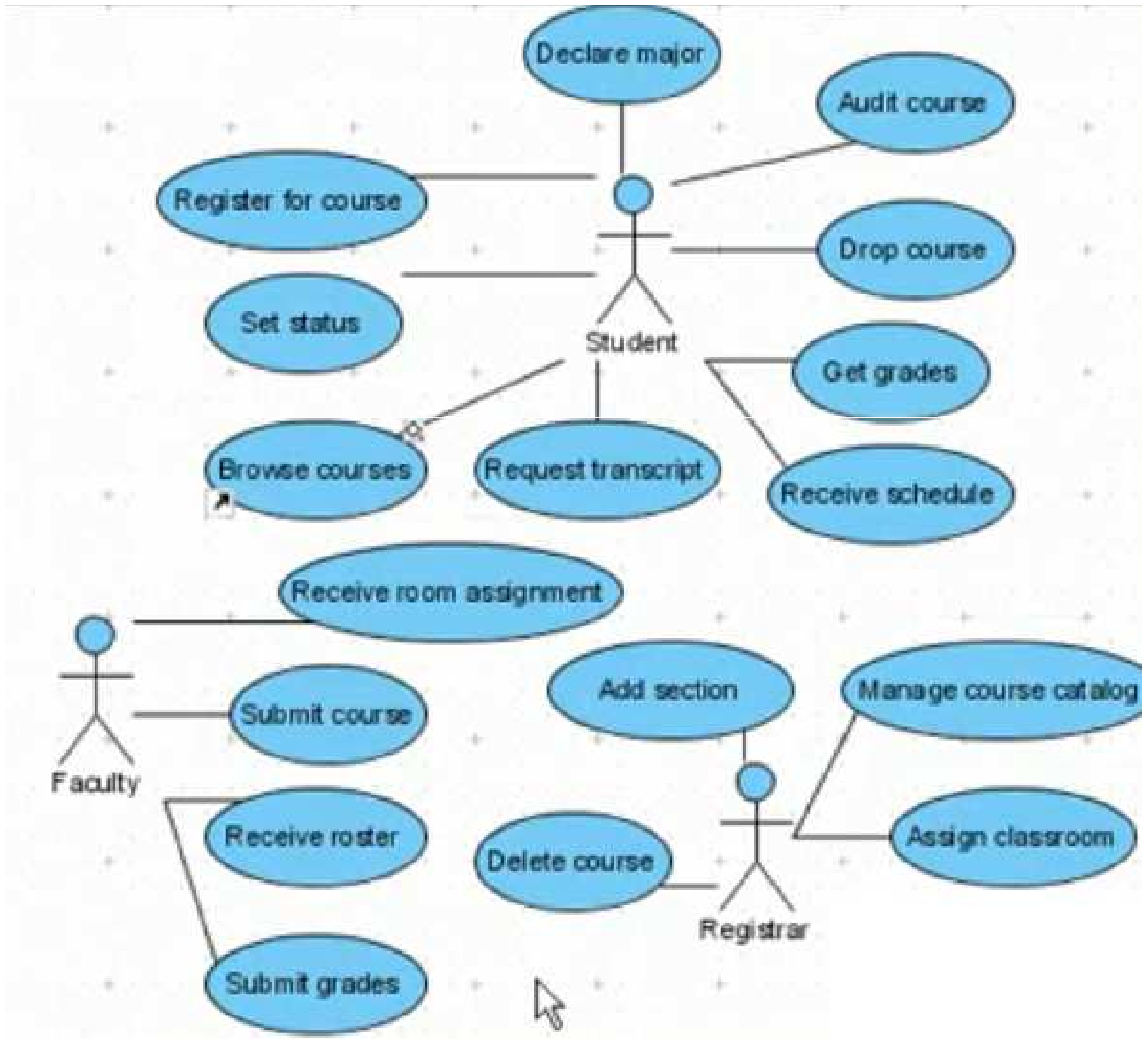




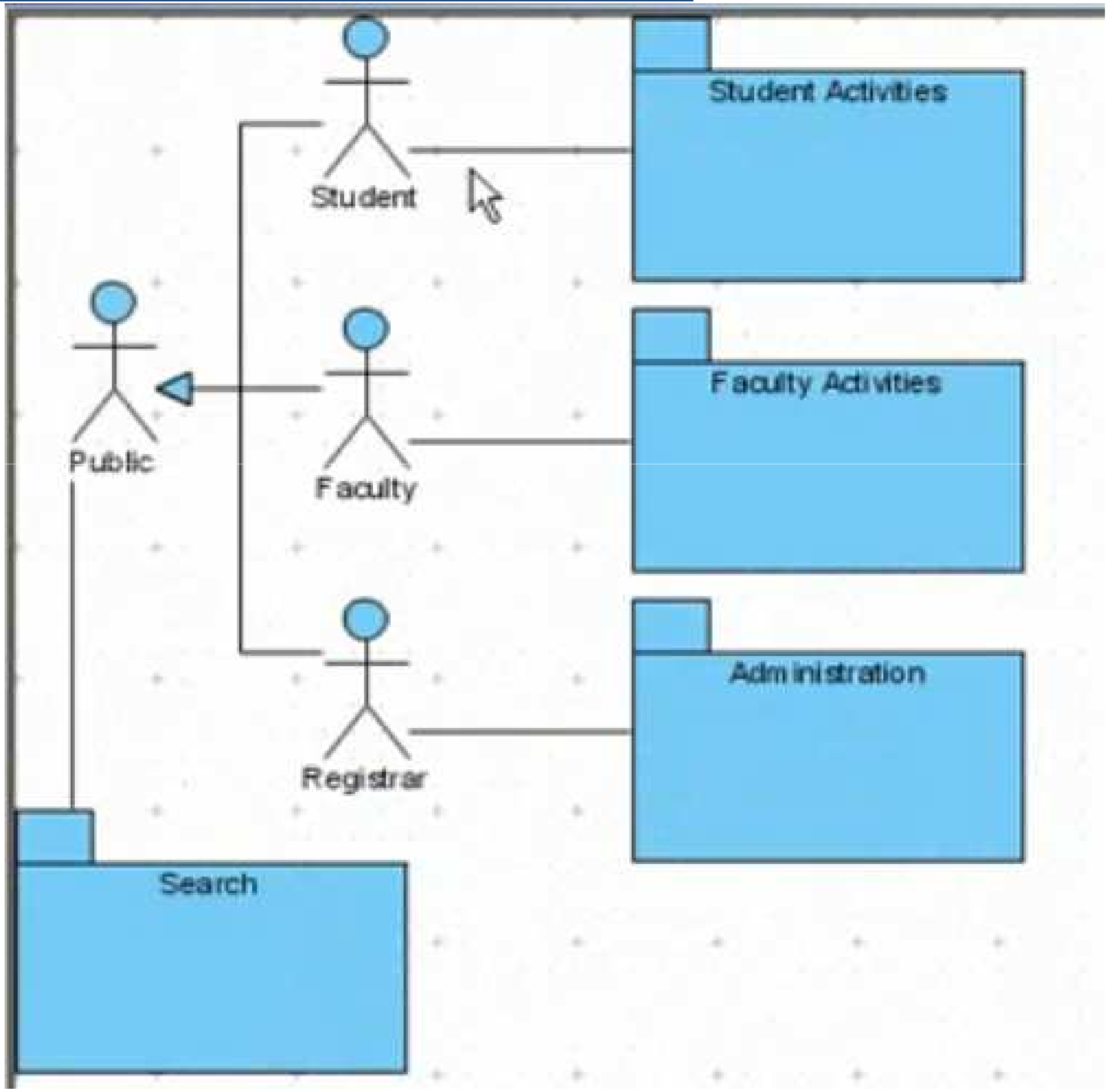
Use case Diagram ➔➔

Use case Package Diagram

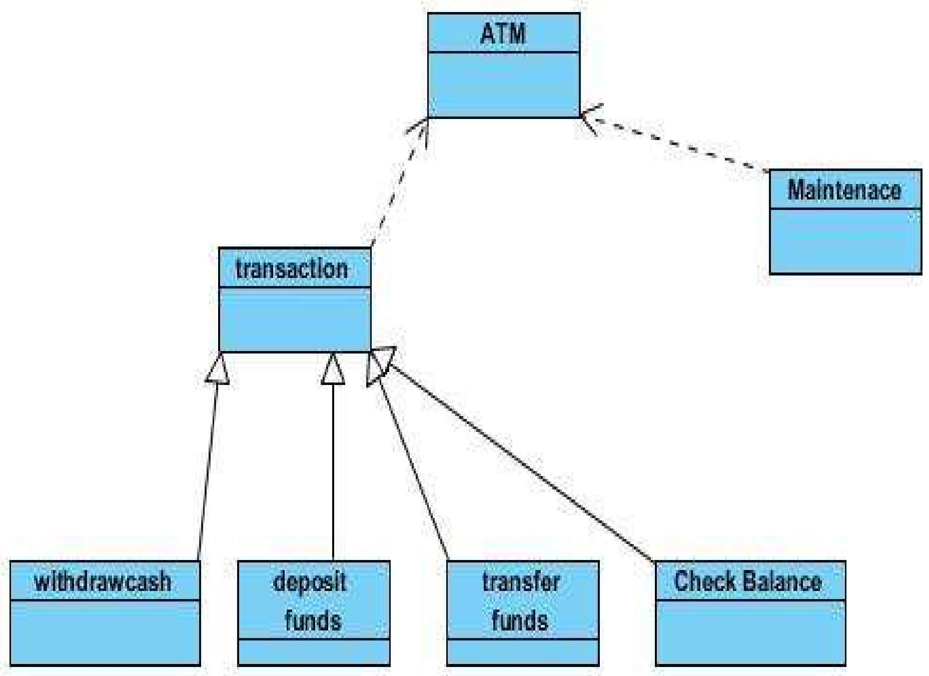
Use Case Package Diagram:

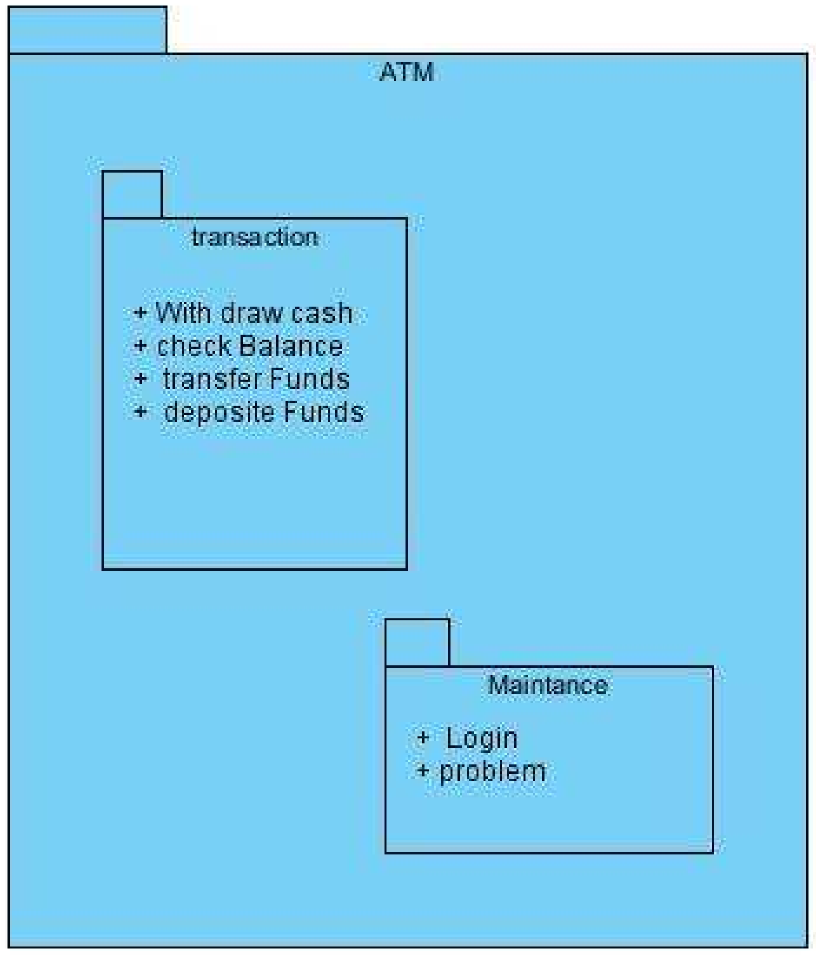


Use Case Package Diagram:



Class Package Diagram





class Diagram ➔➔ class Package Diagram

When to use Package Diagram ???

- A large complex project can hundred of classes. Without some way to organization those classes. It becomes impossible to make sense of tem all.
- Packages create a structure for you classes or other Uml element by grouping related elements.

Use of Package Diagram:

- When you want to show high level view of the system.
- To keep track of dependencies.
- With the large system to show its major element and how they relate to one another.
- To divide a complex system into module
- Package diagrams can use packages that represent the different layers of a software system to illustrate the layered architecture of a software system.

Reference Links:

- http://articles.techrepublic.com.com/5100-10878_11-1045720.html
- http://vapvarun.com/study/softE/john%20wiley%20and%20sons%20-%20programming%20with%20object-oriented%20programming/5399final/lib0147.htm
- http://www.uml-diagrams.org/package-diagrams.html
- http://www.uml-diagrams.org/package-diagrams-examples.html#layered-application-model
- http://www.sparxsystems.com/resources/uml2_tutorial/uml2_packagediagram.html
- http://en.wikipedia.org/wiki/Package_%28UML%29\
- http://en.wikipedia.org/wiki/Package_diagram
- http://www.edrawsoft.com/uml-package.php

Reference Links:

- http://books.google.com/books?id=s1sIlw83-pQC&pg=PA73&lpg=PA73&dq=package+diagram+uml&source=bl&ots=oHf1jpMO04&sig=7-4ngnbNVcDZ0s0b34QMkUIBurQ&hl=en&ei=NGi_TIj3KMOGnQeW6o2KDg&sa=X&oi=book_result&ct=result&resnum=12&ved=0CFMQ6AEwCw#v=onepage&q=package%20diagram%20uml&f=false
- http://dictionary.sensagent.com/package+diagram/en-en/
- http://www.agilemodeling.com/artifacts/packageDiagram.htm
- http://commons.wikimedia.org/wiki/Category:Package_diagrams
- http://www.visual-paradigm.com/VPGallery/diagrams/Package.html
- http://translation.sensagent.com/translate/package%20diagram/en/multilingual.html
- http://www.agilemodeling.com/style/packageDiagram.htm

Questions ??????