UML Package Diagrams

Introduction

- The UML Package Diagram presents separated groups of elements.
- Nearly all UML elements can be grouped into packages.
- Each package has a separated name space. Referring an element that belongs to a specific package from outside of that package must include the package name preceding the element name we try to refer.

Introduction

- Technically we can use the package construct to organize any type of UML elements.
- The package construct is usually used to organize classes in the following cases:

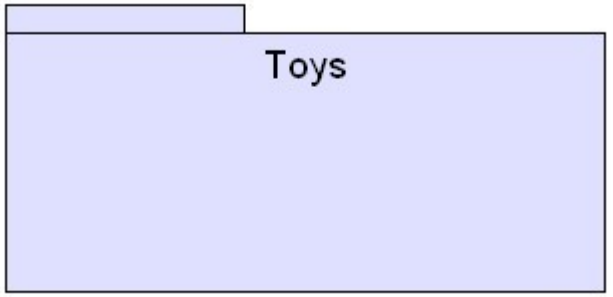
  Classes that belong to the same framework will be placed in the same package.

  Classes in the same inheritance hierarchy usually belong to the same package.

  Classes that have the aggregation / composition relationship with each other usually belong to the same package.

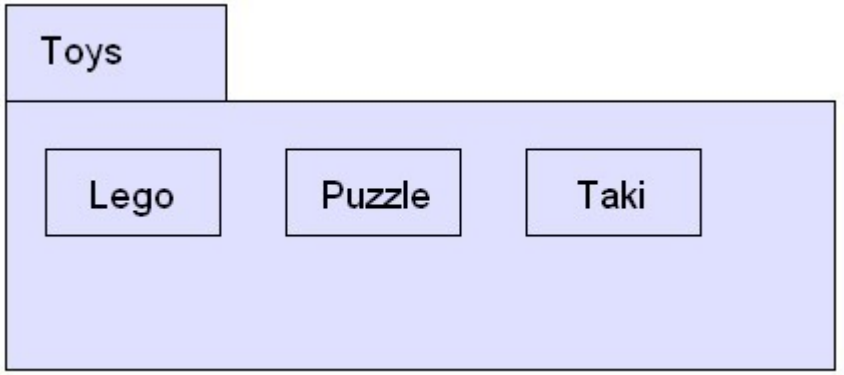
Package Representation

- Depicting a package is done using a rectangle that has a tab attached to its top left.



Package Representation

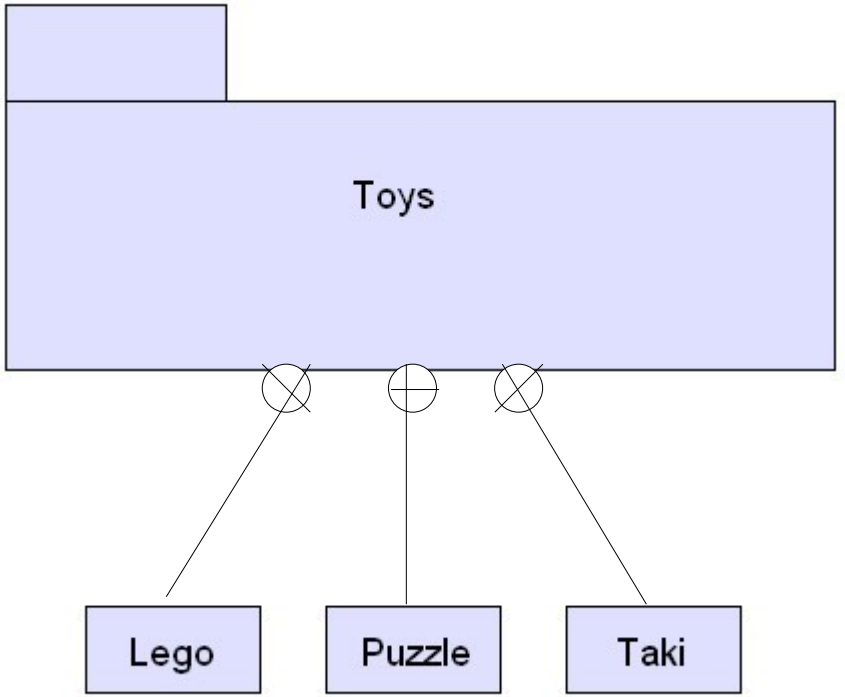
- Within the package we can draw the elements it includes.
- When doing so, it is possible to write the package name within the package top left tab.



Package Representation

- Alternatively, it is possible to draw each one of the elements outside of the package area and connect each one of them with the package using a solid line and a small circle with a plus sign in it at the end nearest the package.

Package Representation



Package Elements Visibility

- Each element within a package can have one of the following two visibility levels:

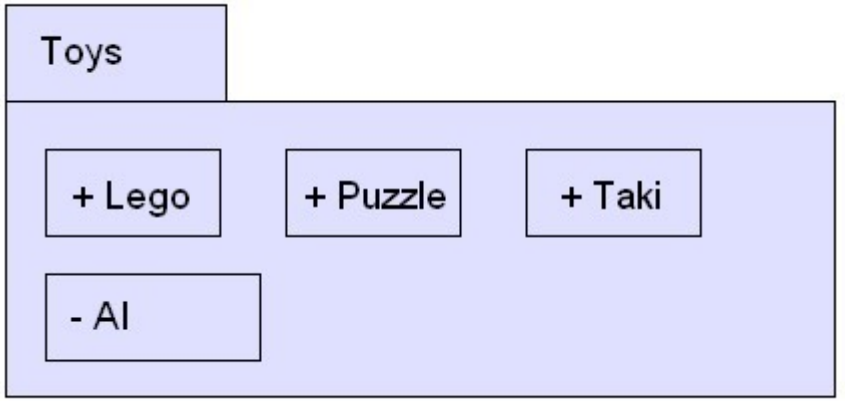
Private

The element can be used by other elements that belong to the same package only. A private element will be marked with '-'.

Public

The element can be used by all elements from all packages. A public element will be marked with '+'.

Package Elements Visibility



Importing Packages

- In order to avoid the full qualified names when accessing elements that belong to another package, UML allows importing one package to another.
- When a package is imported to another package, all elements of the imported package are accessible without having the need to use the full qualified name.
- Done by drawing a dashed line between the two packages and an open arrow pointing at the imported package.

Importing Packages

Importing Packages

- By default, when a package imports another package, all elements that belong to the imported package are given a public visibility within the importing package. As a result of that, if a third package imports the importing package all elements that belong to the first imported package are now accessible both from within the first importing package and from within the second one.

Importing Packages

All public elements that belong to AI Modules become public within Games. As a result of that, given that Mobile Games import Games, the AI Modules elements are accessible both from Games and from Mobile Games packages. Private members cannot be imported.

Accessing Packages

- If using <<access>> instead of <<import>> then elements that belong to a package that another package now accesses get the private accessibility, which means that a third package that imports from the accessing package won't be able to use these elements.

Accessing Packages

All private elements that belong to AI Modules remain private elements that belong to AI Modules. None of these elements is copied to Games. None of these elements can be accessed from within Mobile Games or from Games. The public elements that belong to AI Modules become accessible for Games as private ones only. They aren't accessible from Mobile Games.

Merging Packages

- When two packages merge with each other then elements with the same name are merged. The elements that belong to the package that performs the merge with the other package get a generalization relationship with the other class.
- Private members from any given package are not merged with any other package.
- If the two merged packages include a sub package with the same name a merge process will start between the two sub packages as well.

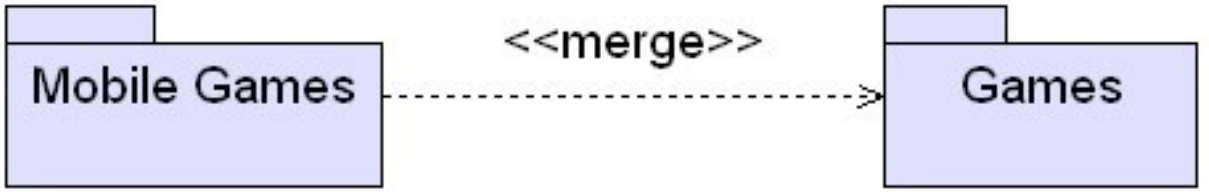
Merging Packages

- Referencing the original elements in the merged package is still feasible by using the full qualified name.

  e.g. Assuming that package ComputerProtocols merges with HumanProtocols reference the original elements of HumanProtocols is still feasible using the full qualified name that includes the HumanProtocols package name.

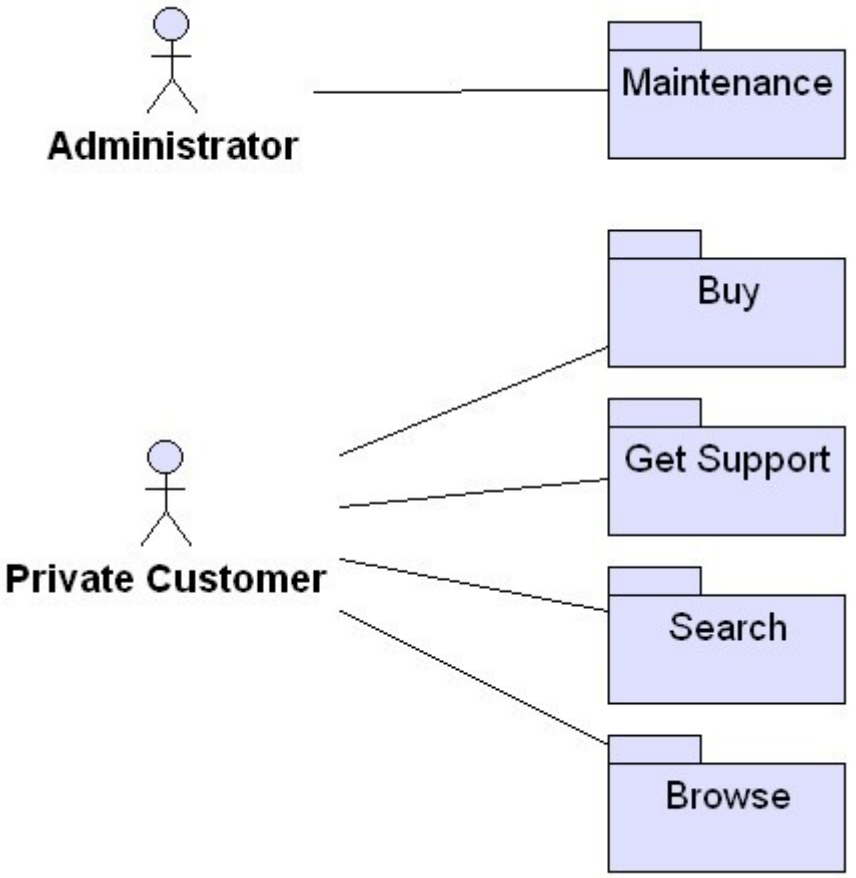
- Elements and sub packages that exist only in one of the two merging packages will remain unchanged.
- Any import from the merged package becomes an import from the merging package.

Merging Packages



Use Case Packages

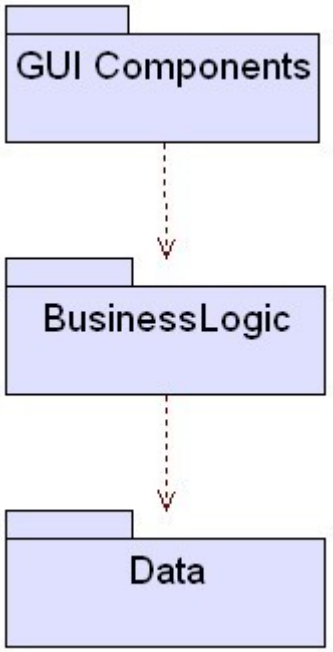
- Taking use cases and placing them in packages can improve the clarity of the use case diagram.



Packages Dependencies Graph

- Having one diagram that includes all packages that we plan to have in our system together with dashed open arrow lines connecting between them and representing the direct dependencies can assist us planning our project development in a more efficient way.
- Having all dependencies flowing in one direction is a good sign for less complexity during the developing process.

Packages Dependencies Graph



UML Package Diagrams

Introduction

- The UML Package Diagram presents separated groups of elements.
- Nearly all UML elements can be grouped into packages.
- Each package has a separated name space. Referring an element that belongs to a specific package from outside of that package must include the package name preceding the element name we try to refer.

Assuming that we set up the “Toys” package and inside that package we have the “Lego” element. In order to refer “Lego” from outside of the “Toys” package we will need to use the full qualified name Toys::Lego

Introduction

- Technically we can use the package construct to organize any type of UML elements.
- The package construct is usually used to organize classes in the following cases:

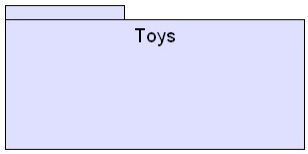
  Classes that belong to the same framework will be placed in the same package.

  Classes in the same inheritance hierarchy usually belong to the same package.

  Classes that have the aggregation / composition relationship with each other usually belong to the same package.

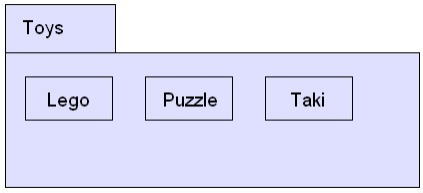
Package Representation

- Depicting a package is done using a rectangle that has a tab attached to its top left.



Package Representation

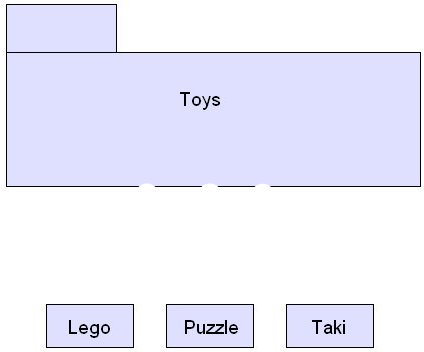
- Within the package we can draw the elements it includes.
- When doing so, it is possible to write the package name within the package top left tab.

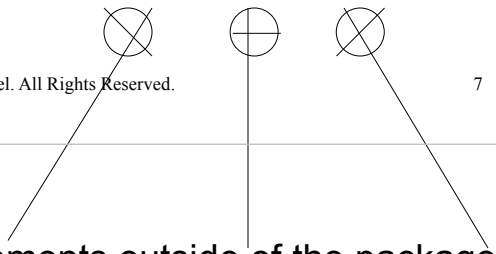


Package Representation

- Alternatively, it is possible to draw each one of the elements outside of the package area and connect each one of them with the package using a solid line and a small circle with a plus sign in it at the end nearest the package.

Package Representation





The advantage drawing the elements outside of the package area is the possibility to draw these elements with more details.

Package Elements Visibility

- Each element within a package can have one of the following two visibility levels:

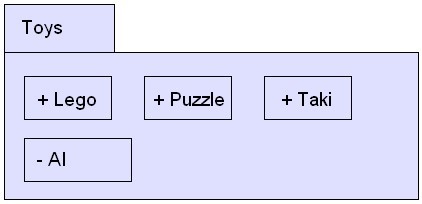
Private

The element can be used by other elements that belong to the same package only. A private element will be marked with '-'.

Public

The element can be used by all elements from all packages. A public element will be marked with '+'.

Package Elements Visibility



All elements within the Toys package have the public accessibility level except for AI. The AI element is private. Therefore, only the elements within Toys package can use it.

Importing Packages

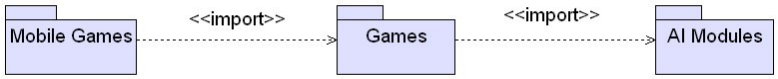
- In order to avoid the full qualified names when accessing elements that belong to another package, UML allows importing one package to another.
- When a package is imported to another package, all elements of the imported package are accessible without having the need to use the full qualified name.
- Done by drawing a dashed line between the two packages and an open arrow pointing at the imported package.

Importing Packages

Importing Packages

- By default, when a package imports another package, all elements that belong to the imported package are given a public visibility within the importing package. As a result of that, if a third package imports the importing package all elements that belong to the first imported package are now accessible both from within the first importing package and from within the second one.

Importing Packages



All public elements that belong to AI Modules become public within Games. As a result of that, given that Mobile Games import Games, the AI Modules elements are accessible both from Games and from Mobile Games packages. Private members cannot be imported.

Accessing Packages

- If using <<access>> instead of <<import>> then elements that belong to a package that another package now accesses get the private accessibility, which means that a third package that imports from the accessing package won't be able to use these elements.

Accessing Packages

All private elements that belong to AI Modules remain private elements that belong to AI Modules. None of these elements is copied to Games. None of these elements can be accessed from within Mobile Games or from Games. The public elements that belong to AI Modules become accessible for Games as private ones only. They aren't accessible from Mobile Games.

Merging Packages

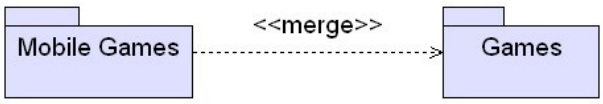
- When two packages merge with each other then elements with the same name are merged. The elements that belong to the package that performs the merge with the other package get a generalization relationship with the other class.
- Private members from any given package are not merged with any other package.
- If the two merged packages include a sub package with the same name a merge process will start between the two sub packages as well.

Merging Packages

- Referencing the original elements in the merged package is still feasible by using the full qualified name.

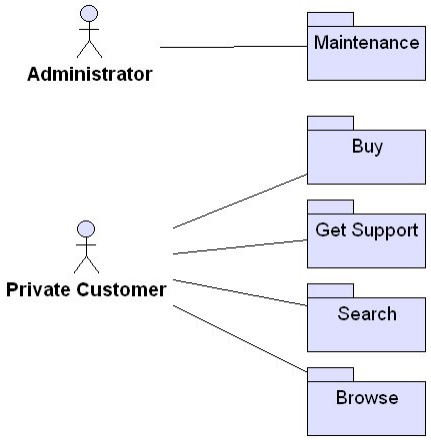
  e.g. Assuming that package ComputerProtocols merges with HumanProtocols reference the original elements of HumanProtocols is still feasible using the full qualified name that includes the HumanProtocols package name.
- Elements and sub packages that exist only in one of the two merging packages will remain unchanged.
- Any import from the merged package becomes an import from the merging package.

Merging Packages



Use Case Packages

- Taking use cases and placing them in packages can improve the clarity of the use case diagram.



Packages Dependencies Graph

- Having one diagram that includes all packages that we plan to have in our system together with dashed open arrow lines connecting between them and representing the direct dependencies can assist us planning our project development in a more efficient way.
- Having all dependencies flowing in one direction is a good sign for less complexity during the developing process.

Packages Dependencies Graph