Component Diagrams & Package Diagrams

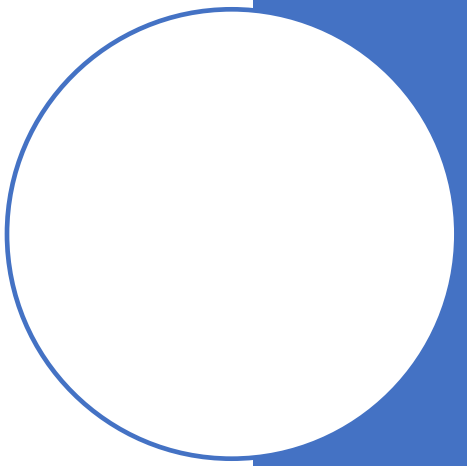
Book: A pragmatic introduction to UML, by Miles and Hamilton.

Managing and Reusing Your System's Parts: Component Diagrams

It's helpful to **plan out the high-level pieces of your system** to establish the architecture and manage complexity and dependencies among the parts.

Components are used to **organize a system into manageable and reusable pieces** of software.

Components diagrams **allow you to identify how your system's parts are organized into modules and components**, and also helps you to manage layers within your system's architecture.



Components

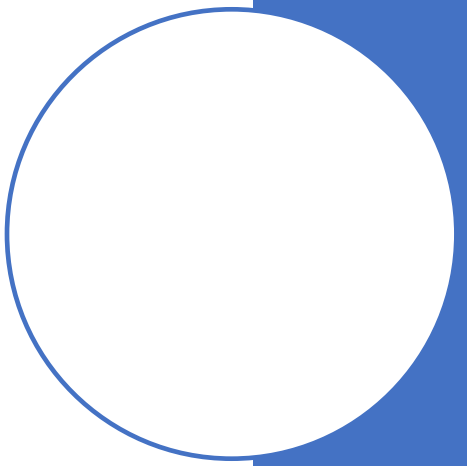
Component: A building block.

Good candidates for components are items that perform a key functionality and will be used frequently throughout your system, i.e., XML parsers, or online shopping carts etc.

In an e-commerce website, there may be a component used to access the data.

Difference between class and components?

To make loosely coupled components, they should be accessed via interfaces.

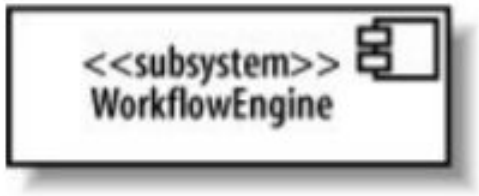


Basic component in UML

The basic component symbol showing a ConversionManagement component

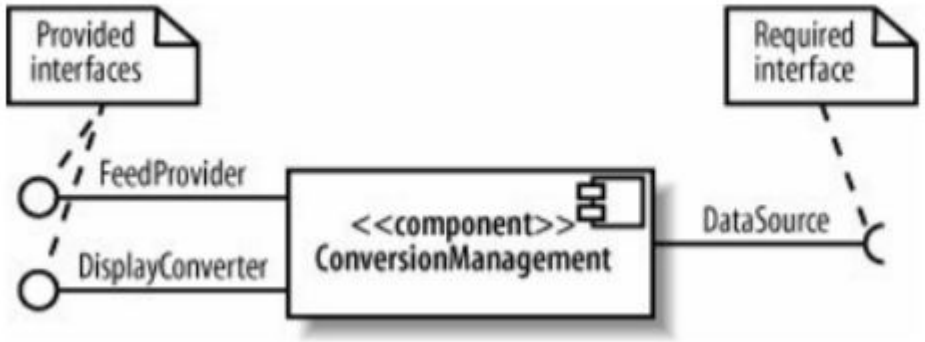
Component used in the CMS that converts blogs to different formats.

Subsystem



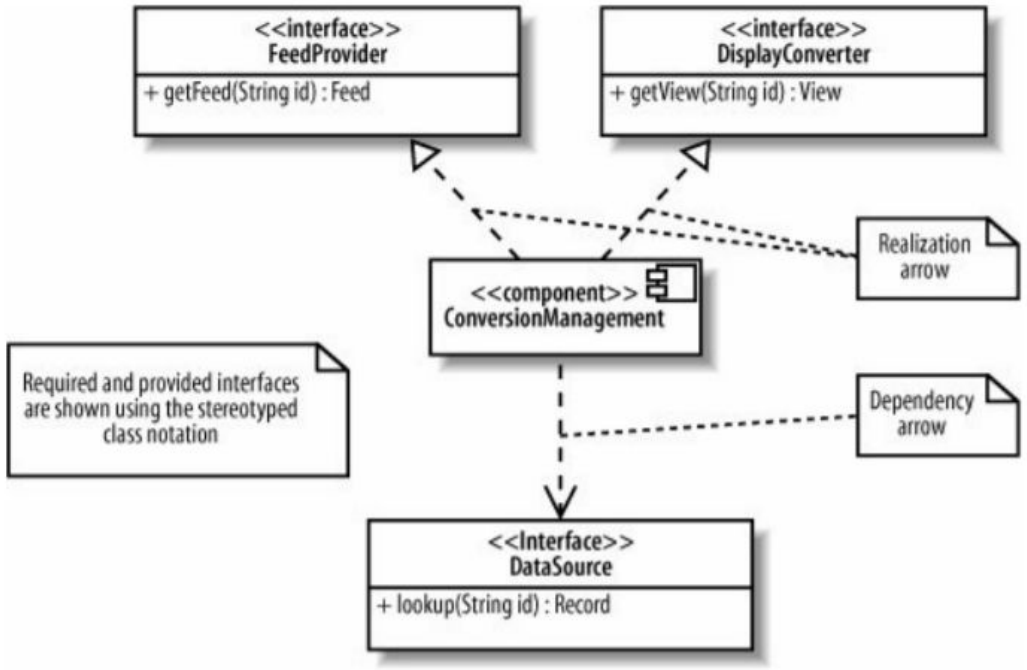
Provided and Required Interface
Ball and socket notation

The ball and socket notation for showing a component's provided and required interfaces



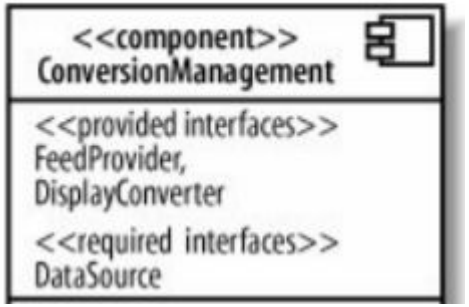
This figure shows that the ConversionManagement component provides the FeedProvider and DisplayConverter interfaces and requires the DataSource interface.

Another Representation Using Class Notation



If a component realizes an interface, draw a **realization arrow** from the component to the interface. If a component requires an interface, draw a **dependency arrow** from the component to the interface

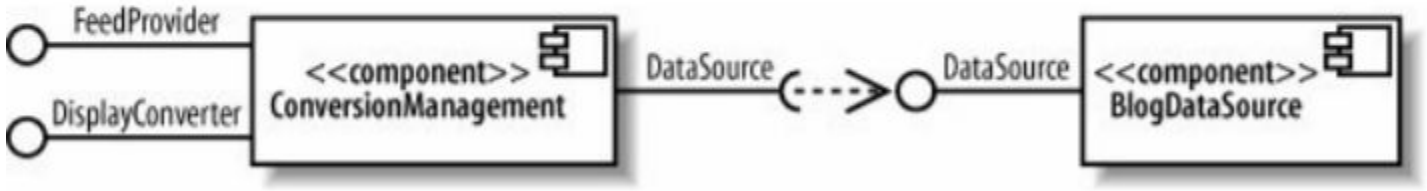
More Compact Way



The most compact way of showing required and provided interfaces is to list them inside the component.

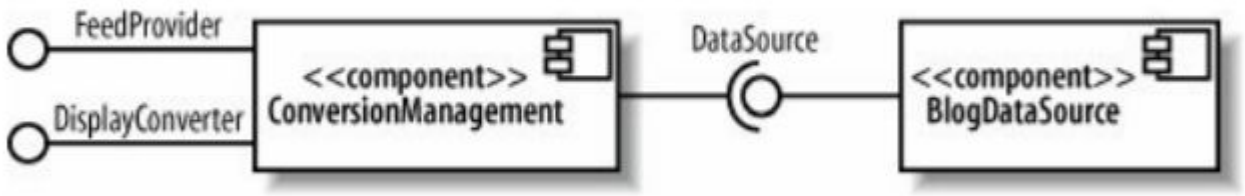
Showing Components Working Together

. The ConversionManagement component requires the DataSource interface, and the BlogDataSource component provides that interface



Showing Components Working Together (Another representation)

Figure 12-8. Presentation option that snaps the ball and socket together

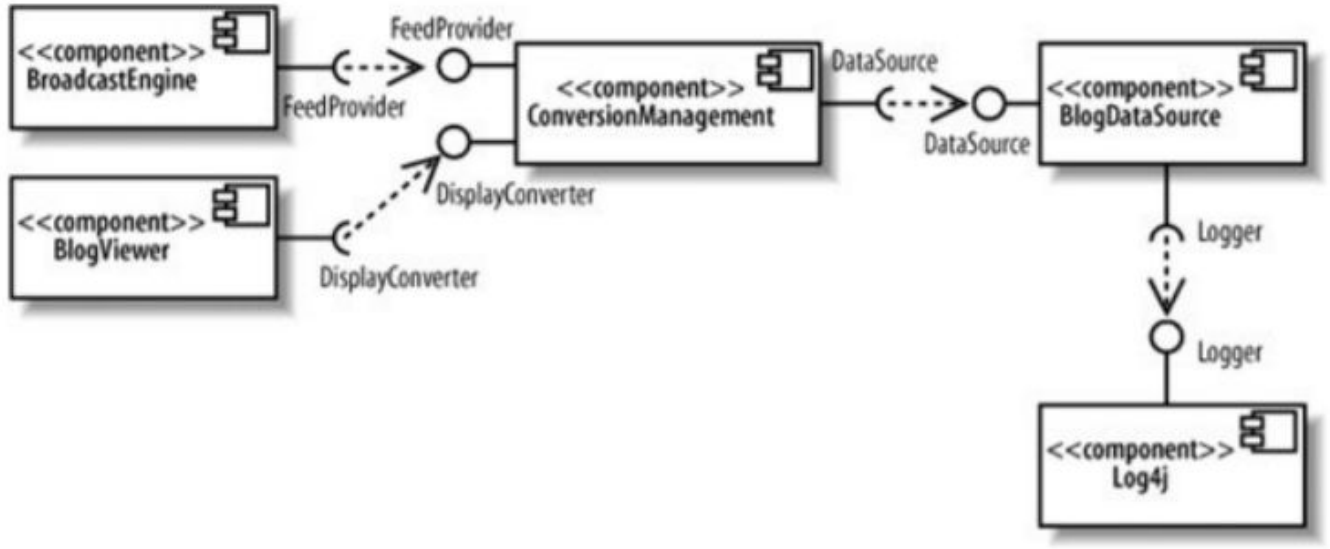


Showing Components Working Together (A more high-level view)

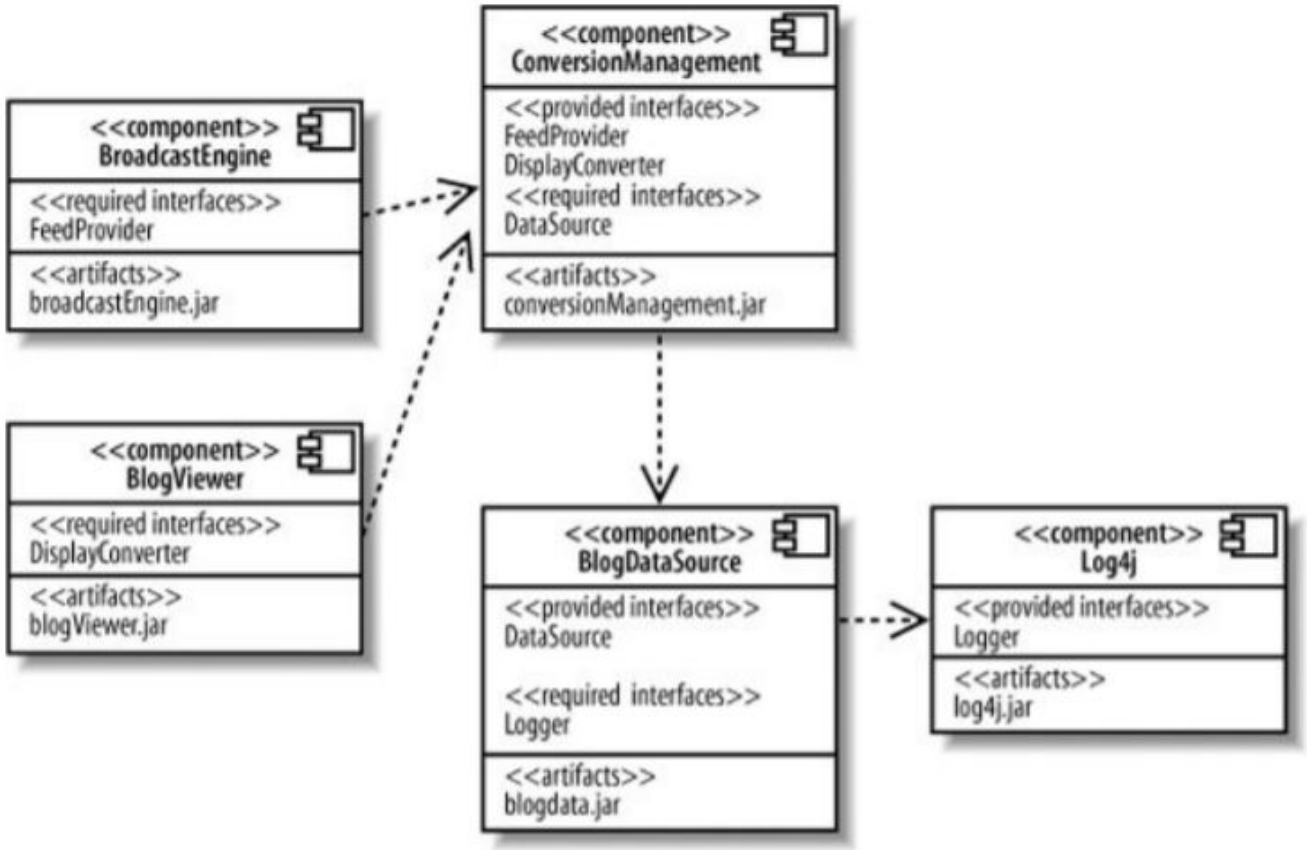
You can draw dependency arrows directly between components to show a higher level view

Multiple Components interaction / Dependencies

Figure 12-10. Focusing on the key components and interfaces in your system

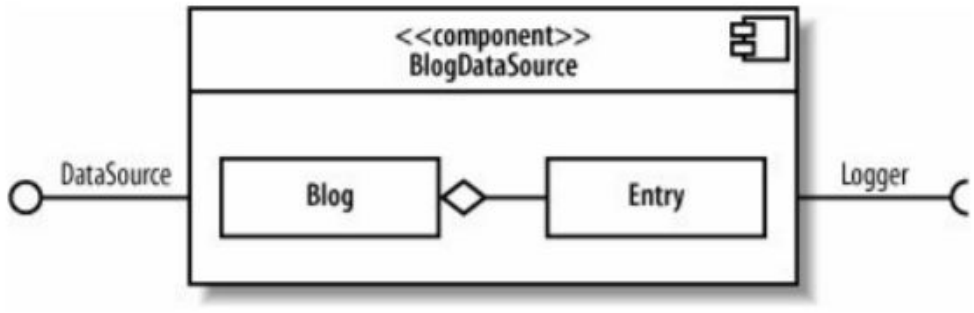


Multiple Components interaction / Dependencies (A simplified higher-level view)



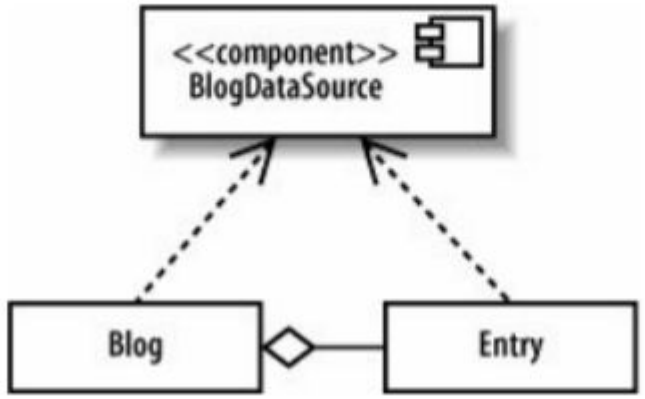
Classes That Realize a Component

Figure 12-12. The Blog and Entry classes realize the BlogDataSource component



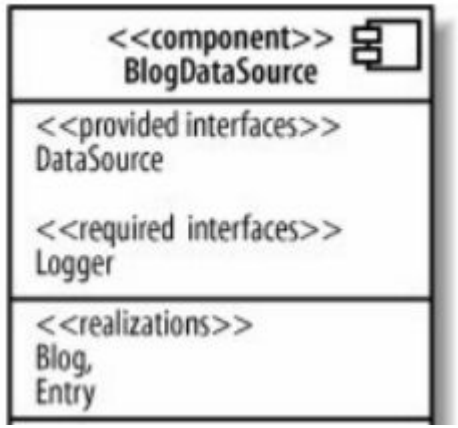
Classes That Realize a Component (Another representation)

Alternate view, showing the realizing classes outside with the dependency relationship



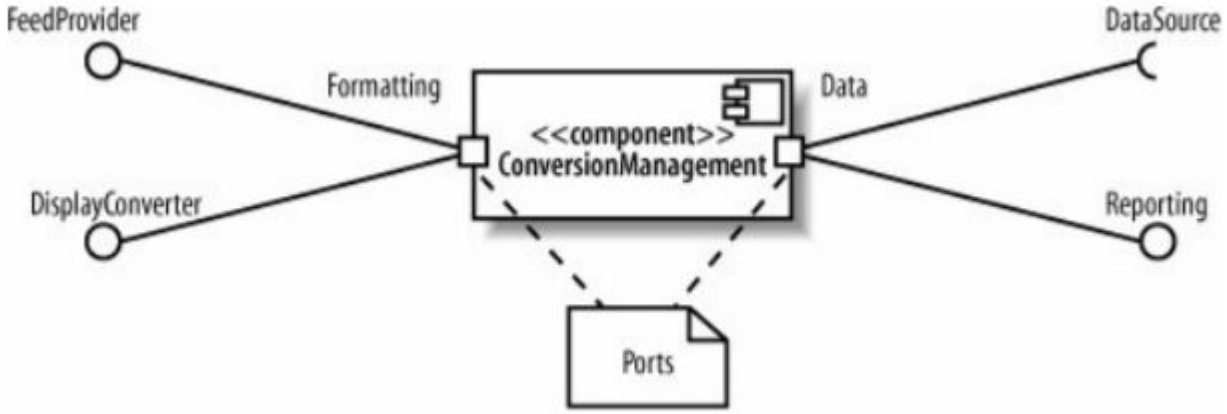
Classes That Realize a Component (Another representation)

Another way to show realizing classes is to list them in a <<realizations>> compartment inside the component

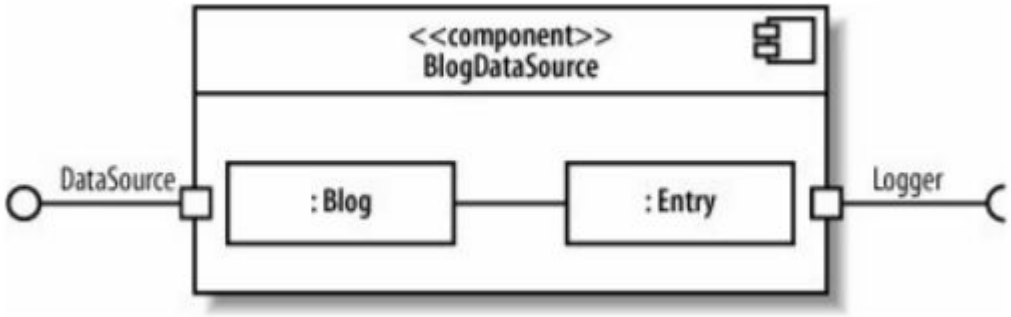


Ports & Internal Structure
Showing Ports

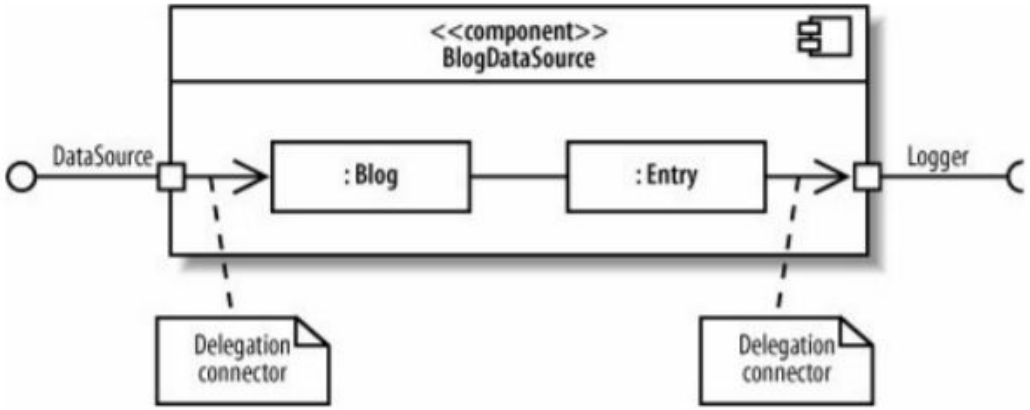
Components can also have ports and internal structure (similar to the ports and internal structure of the class that we studied in the last lecture).



Ports & Internal Structure
Showing Internal Structure

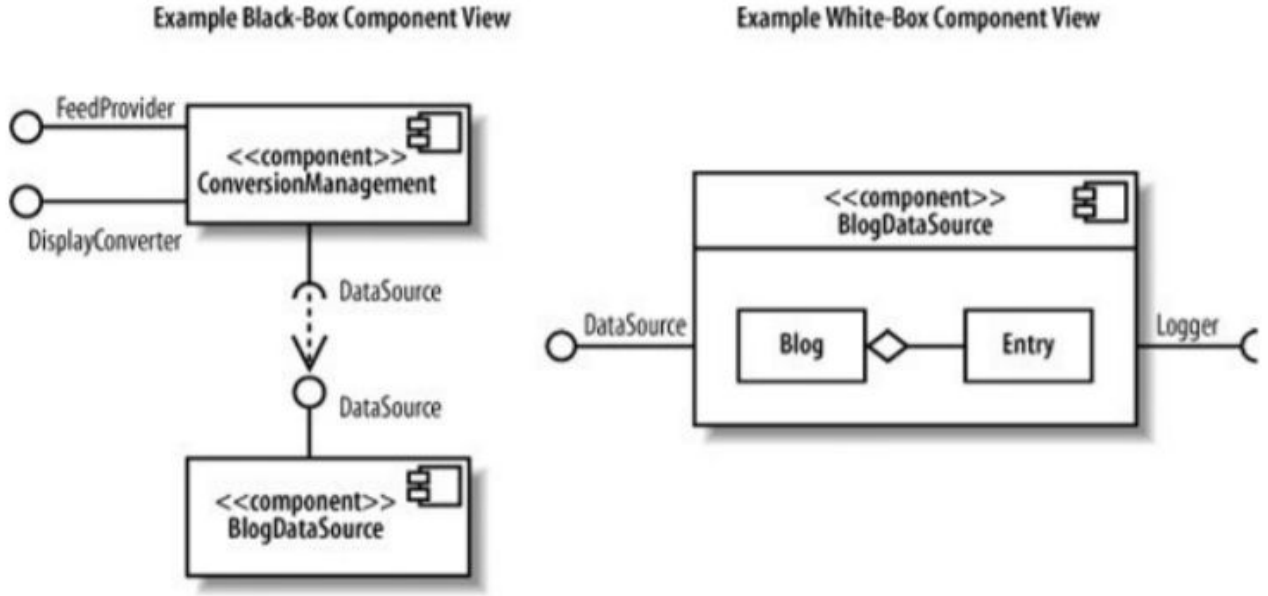


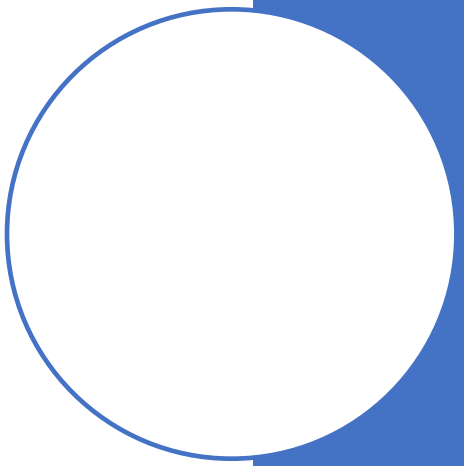
Delegation Connectors



Black-Box and White-Box Component Views

Figure 12-20. Black-box component views are useful for showing the big picture of the components in your system, whereas white-box views focus on the inner workings of a component





Organizing Your Model: Packages

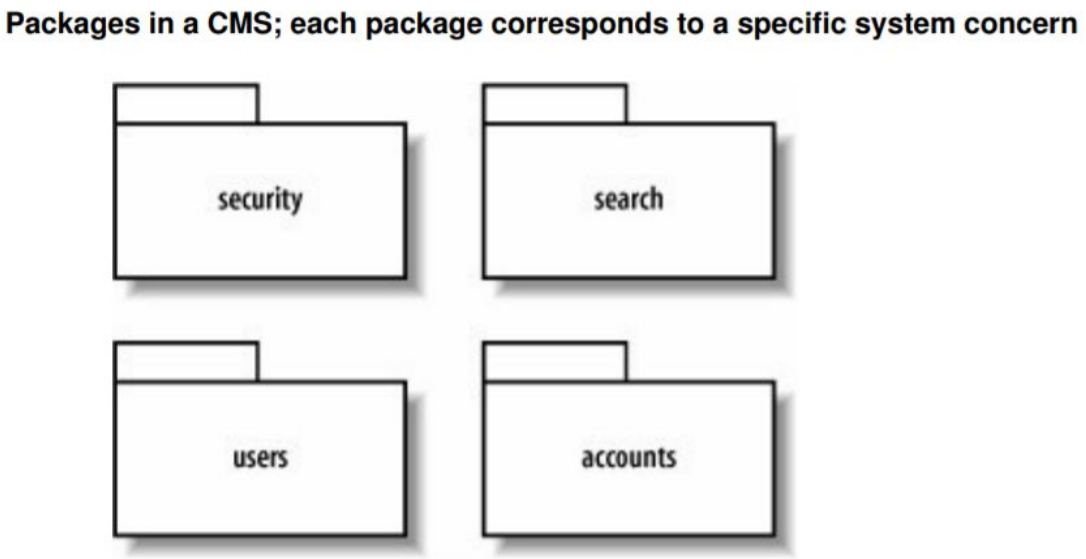
As a software program grows in complexity, it can easily contain hundreds of classes. If you're a programmer working with such a class library, how do you make sense of it?

One way to impose structure is by organizing your classes into logically related groups. Classes concerned with an application's user interface can belong to one group, and utility classes can belong to another.

In UML, groups of classes are modeled with package.

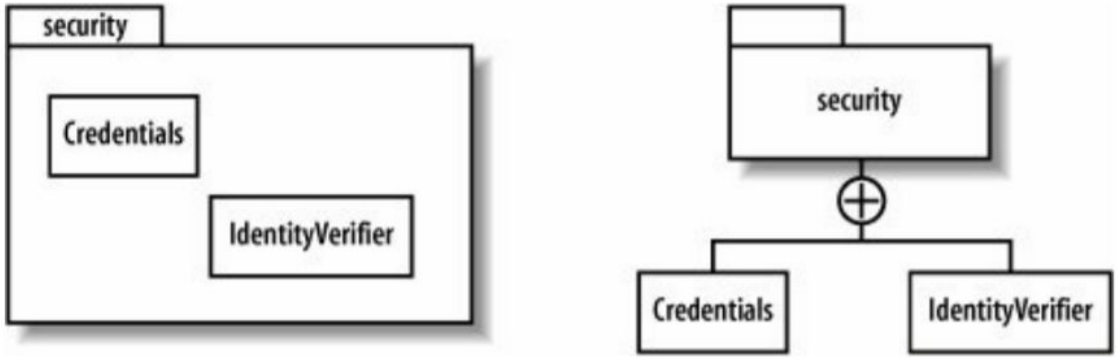
Packages

The name of the package is written inside the folder



Packages

Two ways to show that the Credentials and IdentityVerifier classes are contained in the security package



Packages

Java implementation

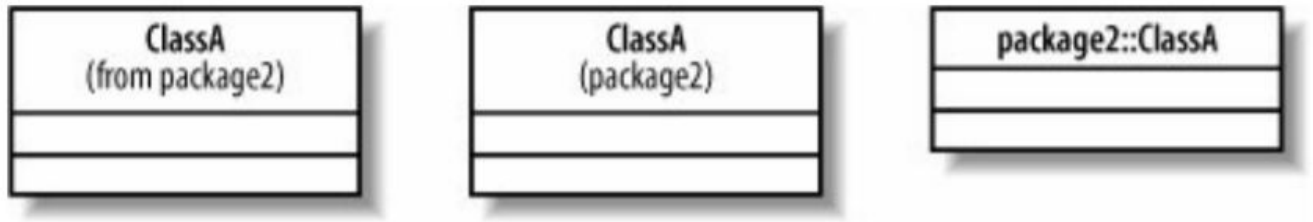
```
package security;

public class Credentials {
    ...
}
```

UML Tool Variation

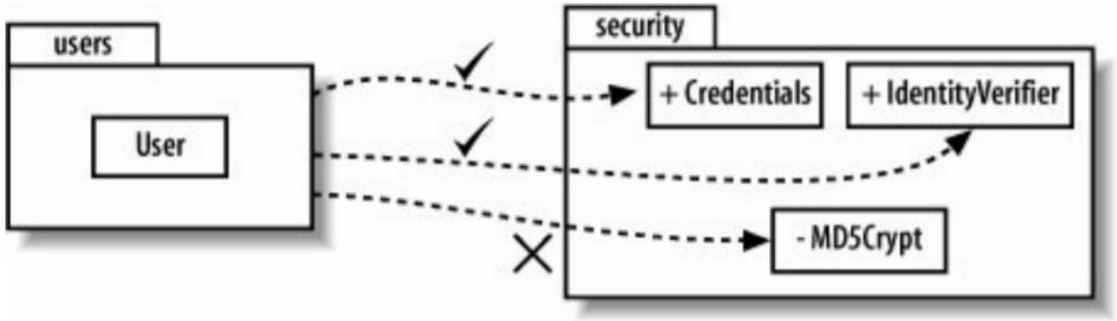
To specify the package that a class belongs to, most UML tools allow you to enter the package name in a class specification dialog or manually drag the class into the package it belongs to in a tree display of the model elements.

Figure 13-7. Common ways UML tools show that a class belongs to a package



Element Visibility

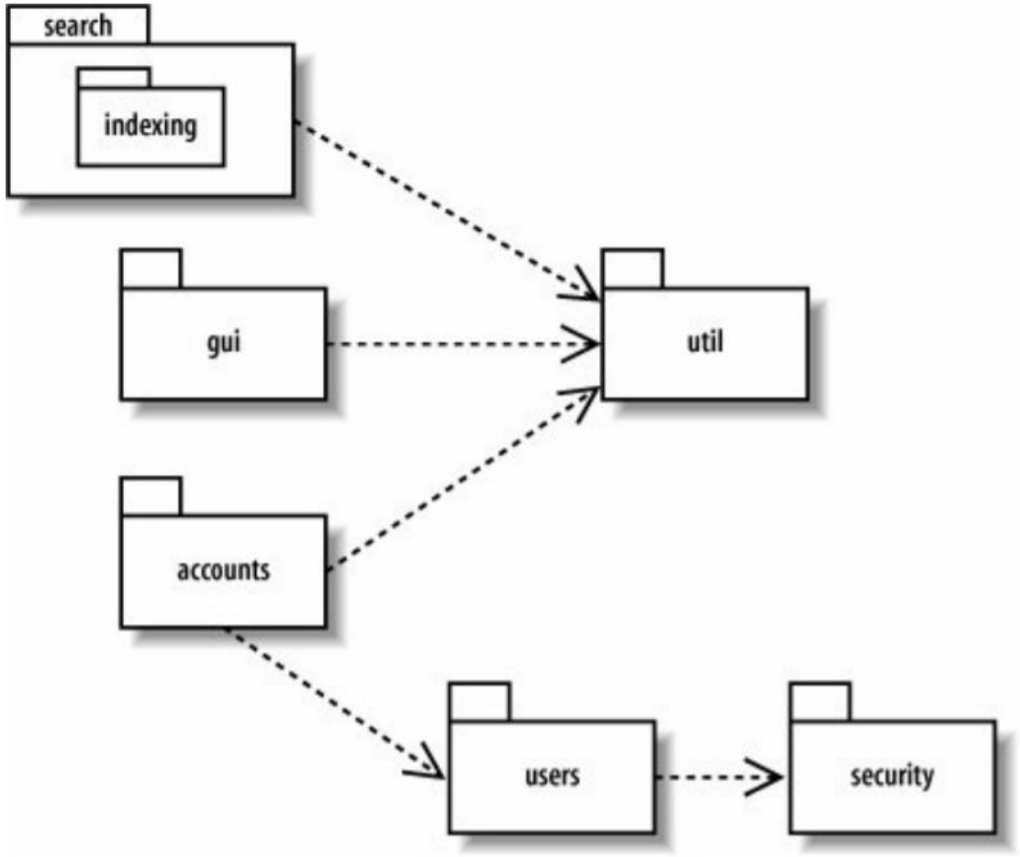
Figure 13-11. Since MD5Crypt has private visibility, it isn't accessible outside the security package



Package Dependency

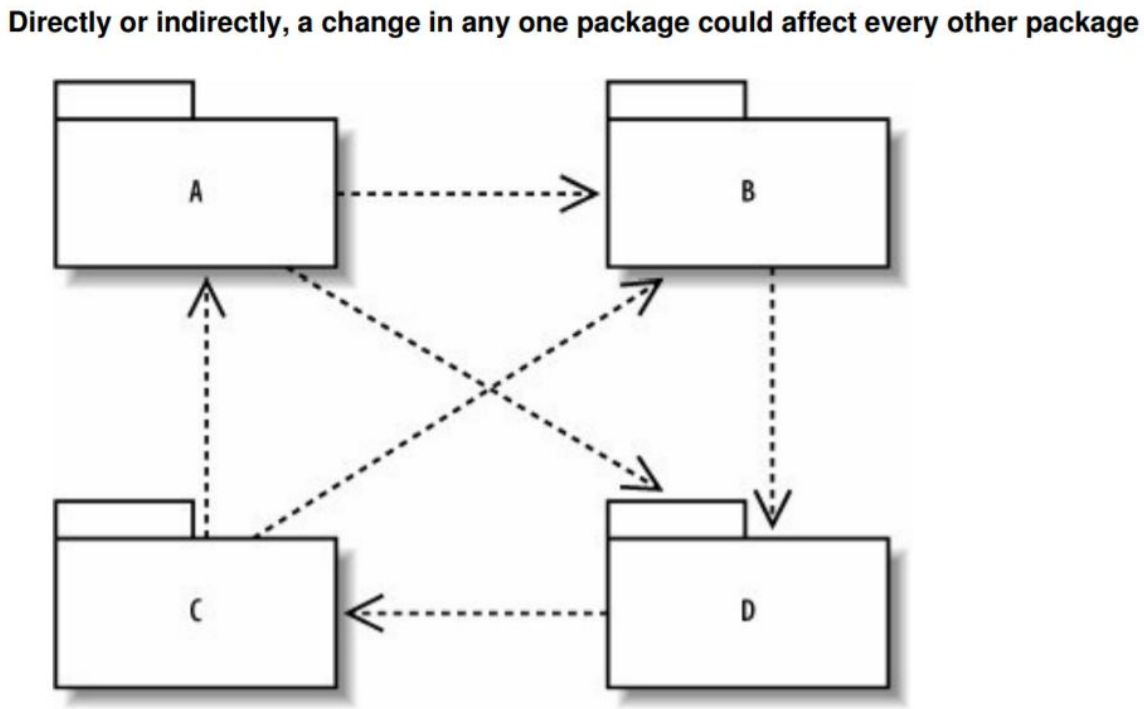
Figure 13-12. Package A depends on package B

Package Dependency



Dependency Disaster

Dependency disaster: a change in any one package could ultimately affect every other package



Using Packages to Organize Use Cases

Packages enable a higher level view of how actors interact with the system