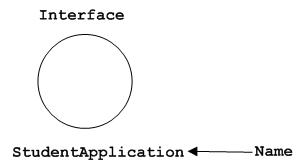
**Some Basic Concepts of OOAD:**

**Interface:**

Interface is represented by a circle as shown below. It has a name which is generally written below the circle.



Interface is used to describe functionality without implementation. Interface is the just like a template where you define different functions not the implementation. When a class implements the interface it also implements the functionality as per the requirement.

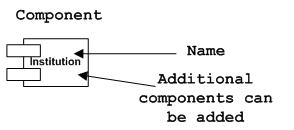
**Component:**

* A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.
* A component is a structured class.
* A component may be

- A source code component

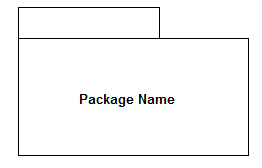
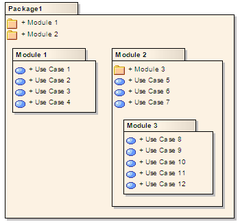
- A run time components or

- An executable component



**Package:**

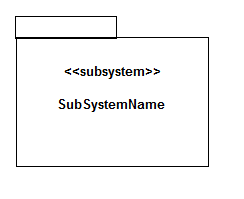
* A package is a general purpose mechanism for organizing elements into groups. i.e. A **package** is a group of similar types of classes, interfaces and sub-packages.
* Package in java can be categorized in two form, built-in package and user-defined package.
* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



We can show the name of a package inside of its large rectangle when the package contents are not shown; otherwise, when the package contents are shown in the large rectangle, you should show the name of a package inside its tab.

**SubSystem:**

* A system is an organized collection of elements that may be recursively decomposed into smaller subsystems and eventually into nondecomposable primitive elements. A unit of hierarchical decomposition for large systems.
* A subsystem is shown as a package marked with the subsystem keyword.

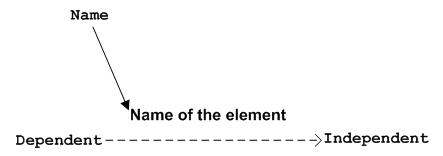
****

**Relationships:**

A model is not complete unless the relationships between elements are described properly. The *Relationship* gives a proper meaning to an UML model. Following are the different types of relationships available in UML.

* Dependency
* Association
* Generalization
* Realization

**Dependency:**

* describes the dependent elements and the direction of dependency.
* Dependency is represented by a dotted arrow as shown below. The arrow head represents the independent element and the other end the dependent element.
* 
* Dependency is used to represent dependency between two elements of a system.

**Association:**

Association is used to represent the relationship between two elements of a system.

2 types:

* + Aggregation
  + Composition

**Association : Aggregation:** A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts

**Association : Composition:** A form of aggregation with strong ownership and coincident life times. The parts cannot survive the whole/aggregate.

**Association: Multiplicity and Navigation:**

* Multiplicity defines how many objects participate in a relationships
  + The number of instances of one class related to ONE instance of the other class
  + Specified for each end of the association
* Associations and aggregations are bi-directional by default, but it is often desirable to restrict navigation to one direction
  + If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

**Association: Multiplicity**

**Example: Multiplicity and Navigation**

**Relationships: Generalization**

* A relationship among classes where one class shares the structure and/or behavior of one or more classes
* Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses
  + Single inheritance
  + Multiple inheritance
* Generalization is an “is-a-kind of” relationship
* **Example: Single Inheritance :** One class inherits from another
* **Example: Multiple Inheritance :** A class can inherit from several other classes

**What Gets Inherited?**

* A subclass inherits its parent’s attributes, operations, and relationships
* A subclass may:
  + Add additional attributes, operations, relationships
  + Redefine inherited operations (use caution!)
* Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

**Realization:**

Realization shows the relationship between an Interface and the class that provides the implementation for the interface.

We can implement this in UML using a closed, hollow arrowhead pointing from the implementation class to the interface with a dashed line.

