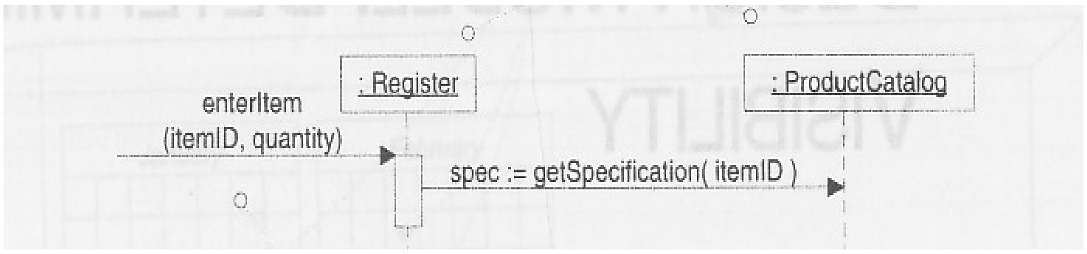
DETERMINING VISIBILITY

1. Introduction

A Visibility is the ability of one object to see or have reference to another. Whenever there is a message passing between two objects, we can determine the visibility between them. For a sender object to send a message to a receiver object, the sender must be visible to the receiver i.e. the sender must have some kind of reference or pointer to the receiver object.



For example, in the interaction diagram above, *ProductCatalog* is visible to *Register*. The *Register* has connection or reference to the *ProductCatalog*, and hence can send message. When creating a design of interacting objects, it is rewuired to ensure that the necessary visibility is present to support message interaction. More generally, it is related to the issue of scope. For instance, the question "Is one resource (such as an instance) within the scope of another?" determines the visibility.

2. Types (scopes) of Visibility

There are four common ways that visibility can be achieved from object A to object B:

1. Attribute visibility - B is an attribute of A.
2. Parameter visibility – B is a parameter of a method of A.
3. Local visibility - B is a (non-parameter) local object in a method of A.
4. Global visibility - B is in some way globally visible.

2.1. Attribute Visibility

Attribute visibility from A to B exists when B is an attribute of A. It is a relatively permanent visibility because it persists as long as A and B exists. This is a very common form of visibility in object-oriented systems.

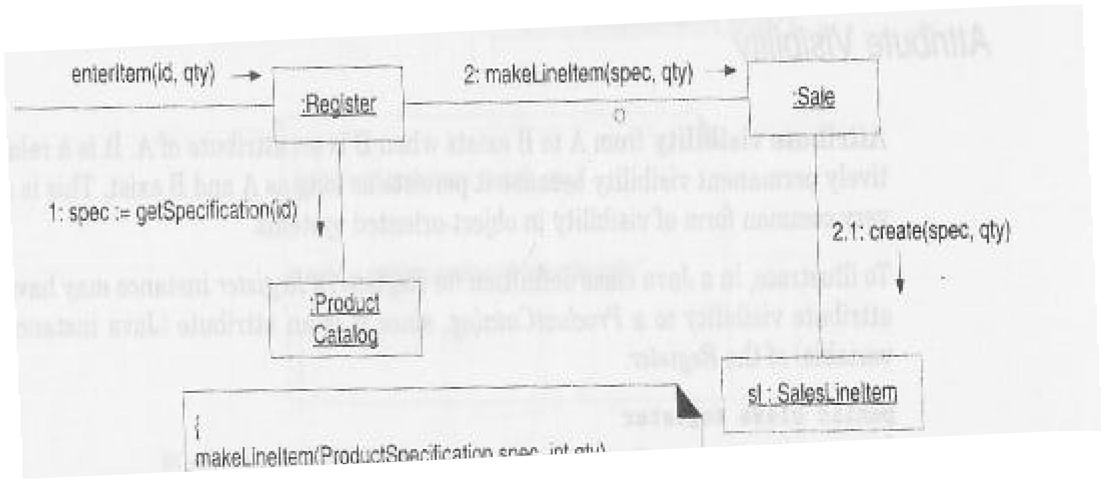
Taking the previous figure as an example, we can obtain attribute visibility and message passing mechanism, for example in Java, through a class definition statements and method creating statements as shown below: -

```
public class Register
{
…
private ProductCatalog catalog:
…
}
```

```
{
public void enterItem (itemID, qty)
{
…
spec = catalog.getSpecification(itemID)
…
}
}
```

2.2. Parameter Visibility

Parameter visibility from A to B exists when B is passed as a parameter to a method of A. It is a relatively temporary visibility because it persists only within the scope of the method.


enterItem(id, qty)
:Register
2: makeLineItem(spec, qty)
:Sale
1: spec := getSpecification(id)
2.1: create(spec, qty)
:Product Catalog
sl : SalesLineItem
{
makeLineItem(ProductSpecification spec, int qty)

In this collaboration diagram, the *makeLineItem* message is sent to a *Sale* instance, passing a ProductSpenification instance as a parameter. Within the scope of the *makeLineltem* method, the Sale has parameter visibility to a *ProductSpecification*.

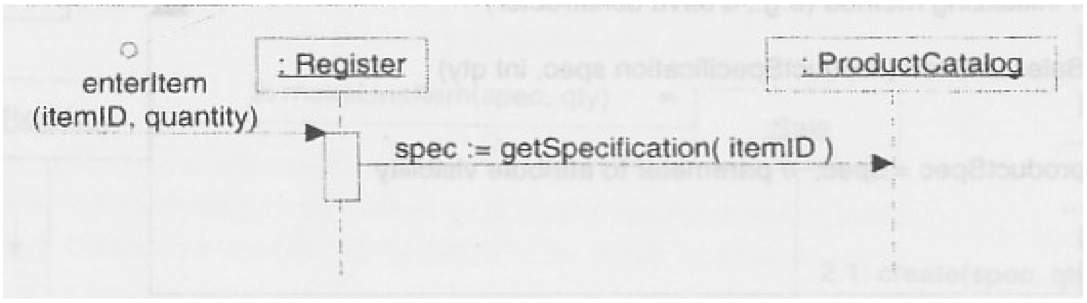
For the visibility, the codes in Java looks something like this: -

```
{
makeLineltem(Productspedfication spec, int qty)
{
…
sl = newSalesLineItem(spec, qty);
…
}
}
```

It is common to transform parameter visibility into attribute visibility. For example, when the Sale creates a new SalesLineItem, it passes a ProductSpecification into its initializing method (in C++ or Java, this would be its constructor). Within the initializing method, the parameter is assigned to an attribute, thus establishing attribute visibility.

2.3. Local Visibility

Local visibility from A to B exists when B is declared as a local object within a method of A. It is a relatively temporary visibility because it persists only within the scope of the method.



Two common means by which local visibility is achieved are

- Create a new local instance and assign it to a local variable.
- Assign the returning object from a method invocation to a local variable. As with parameter visibility, it is common to transform locally declared visibility into attribute visibility.

An example of the second variation (assigning the returning object to a local variable) can be found in the enterItem method of class Register as shown in the figure. An example codes in Java is given below: -

```
{
enterItem (id, qty)
{
…
ProductSpecification spec = catalog.getSpecification(id);
…
}
}
```

2.4. Global Visibility

Global visibility from A to B exists when B is global to A. It is a relatively permanent visibility because it persists as long as A and B exist. It is the least common form of visibility in object-oriented systems.

One way to achieve global visibility is to assign an instance to a global variable, which is possible in some languages, such as C++, but not others, such as Java.

Source:

- Craig Larman, *Applying UMN and Patterns*