

UML

Activity diagram

Class diagram

Classes. Standard representation contains three *compartments*:

1. The *name compartment* (required) contains the class name and other documentation-related information: E.g.:

```
Some_class «abstract»  
  
{ author:      George Jetson  
  modified:    10/6/2999  
  checked_out: y  
}
```

- Guillemets identify **stereotypes**. E.g.: «utility», «abstract» «interface».
- Can use a graphic instead of word. («interface» often represented as small circle)
- Access privileges (see below) can precede name
- Use italics for abstract-class and interface names.

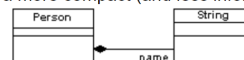
2. The *attributes compartment* (optional):

- *During Analysis*: identify the attributes (i.e. defining characteristics) of the object.
- *During Design*: identify a relationship to a stock class:

This:



is a more compact (and less informative) version of this:



Everything except constant values must be private. Always. Period.

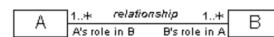
3. The *operations compartment* (optional) contains method definitions:

```
message_name(arguments): return_type
```

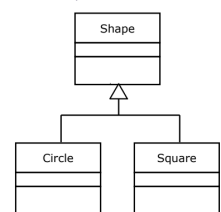
Resist the temptation to use implementation-language syntax.

Visibility (**access privileges**) indicated as follows:¹

+ public
protected



inheritance



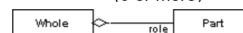
Associations (relationships between classes).

- Associated classes are connected by lines.
- The **relationship** is identified, if necessary, with a < or > to indicate direction (or use solid arrowheads).
- The role that a class plays in the relationship is identified on that class's side of the line.
- Stereotypes (like «friend») are appropriate.
- Unidirectional message flow can be indicated by an arrow (but is implicit in situations where there is only one role):

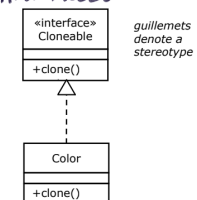


- Cardinality:

1 (usually omitted if 1:1)
n (unknown at compile time, but bound)
0..1 (1..2 1..n)
1..* (1 or more)
* (0 or more)



interfaces



guillemets denote a stereotype

Aggregation (comprises) relationship relationship.¹ Destroying the "whole" does not destroy the parts.



Composition (has) relationship.¹ The parts are destroyed along with the "whole."

Guidelines

Modularity:

— increase cohesion

- class has a clear specific responsibility

— reduce coupling

- class is not connected to or knows too many others

separate the layers

- identify entity, control, and boundary objects
- allow replacing layers

if it does a bunch of different things, it has low cohesion.

coupling is dependency. Too much dependence on majority problems in one method