# INSTANCES

- Instance is a concrete manifestation of an abstraction to which a set of operations can be applied and which has a state that stores the effects of the operations.
- instances and objects are almost same (we cannot say an object of association, it is an instance of association called link)
- an instance is rendered(shown) by underlining its name
- Graphical representation for instances – named instances, as well as anonymous Figure 1.
- Instances don't stand alone; they are tied to an abstraction
- instances can be of classes, components, nodes, use cases, and associations
- Every instance must have a name that distinguishes it from other instances within its context. Figure 2.
- an instance can be used to things dynamically eg:- calling an operation using a class instance.
- an instance/ object can have a state -> it's static properties along with current(dynamic) values it holds. so when you visualize its state, you are really specifying the value of its state at a given moment in time and space. Figure 3.
- Active objects are represented as shown in Figure 4.

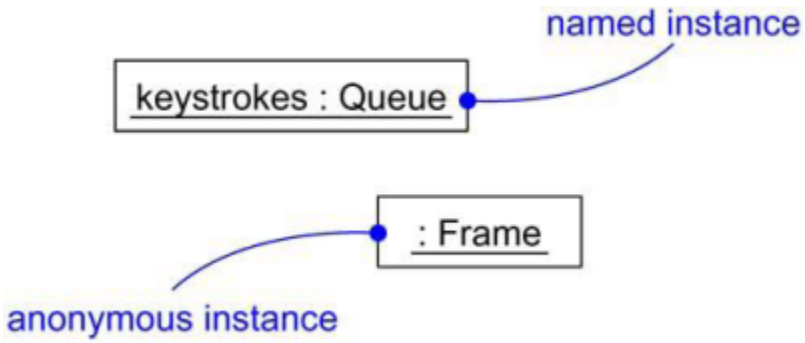


Figure1: instances

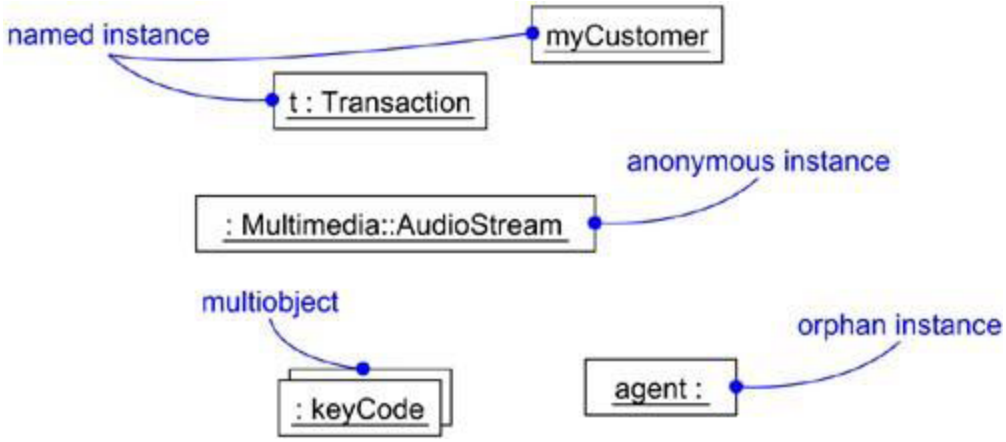


Figure: 2 Named, Anonymous, Multiple, and Orphan Instances

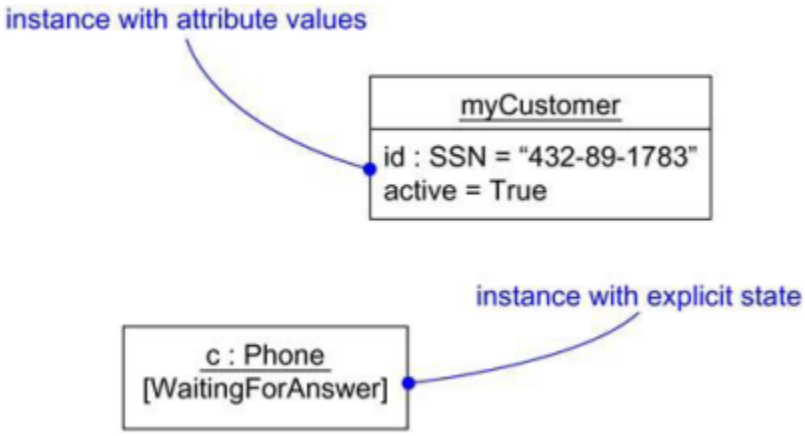


Figure 3 Object or instnace State

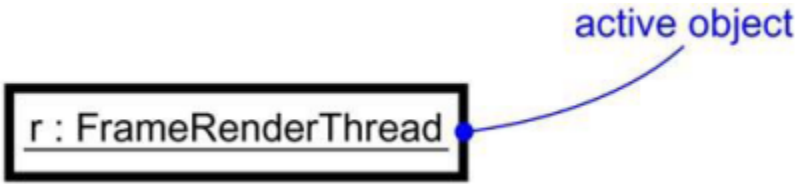


Figure 4 Active Objects

two standard stereotypes that apply to the dependency relationships among objects and among classes:

1. instanceOf – Specifies that the client object is an instance of the supplier classifier
2. instantiate – Specifies that the client class creates instances of the supplier class

two stereotypes related to objects that apply to messages and transitions:

1. become – Specifies that the client is the same object as the supplier, but at a later time and with possibly different values, state, or roles

2. copy – Specifies that the client object is an exact but independent copy of the supplier

standard constraint that applies to objects:

1.transient – Specifies that an instance of the role is created during execution of the enclosing
2.interaction – but is destroyed before completion of execution

**Modeling Concrete Instances**

To model concrete instances,

- Identify those instances necessary and sufficient to visualize, specify, construct, or document the problem you are modeling.
- Render these objects in the UML as instances. Where possible, give each object a name. If there is no meaningful name for the object, render it as an anonymous object.
- Expose the stereotype, tagged values, and attributes (with their values) of each instance necessary and sufficient to model your problem.
- Render these instances and their relationships in an object diagram or other diagram appropriate to the kind of the instance.

Figure 5 shows an object diagram drawn from the execution of a credit card validation system

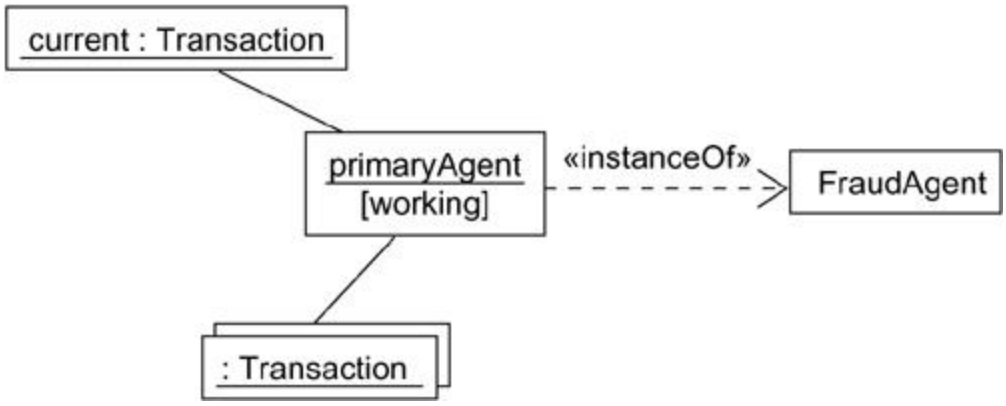


Figure 5 Modeling Concrete Instances

**Modeling Prototypical Instances**

To model prototypical instances,

- Identify those prototypical instances necessary and sufficient to visualize, specify, construct, or document the problem you are modeling.
- Render these objects in the UML as instances. Where possible, give each object a name. If there is no meaningful name for the object, render it as an anonymous object.
- Expose the properties of each instance necessary and sufficient to model your problem.
- Render these instances and their relationships in an interaction diagram or an activity diagram.

Figure 6 shows an interaction diagram illustrating a partial scenario for initiating a phone call in the context of a switch. There are four prototypical objects: a (a CallingAgent), c (a Connection), and t1 and t2 (both instances of Terminal). All four of these objects are prototypical; all represent conceptual proxies for concrete objects that may exist in the real world.

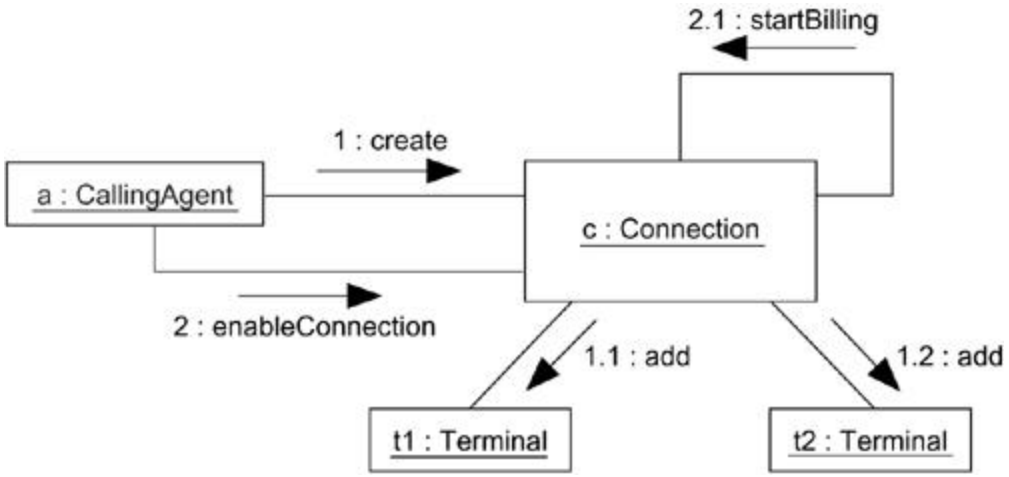


Figure 6 Modeling Prototypical Instances

Source : http://praveenthomasln.wordpress.com/2012/04/01/instances-s8-cs/