ADVANCED CLASSES II

Abstract, Root, Leaf and Polymorphic Elements

Abstract classes are those that do not have any direct instances and is specified in UML by writing its name in italics. A leaf class is a class that have no children and is specified in UML by writing the property leaf below the class's name. A root class is a class that has no parents and is specified in UML by writing the property root below the class's name.

An operation is polymorphic if it is specified with the same signature at different places in the hierarchy of classes. Which operation to invoke is done polymorphically, that is a match is determined at run time according to the type of the object. These are indicated in Figure:4

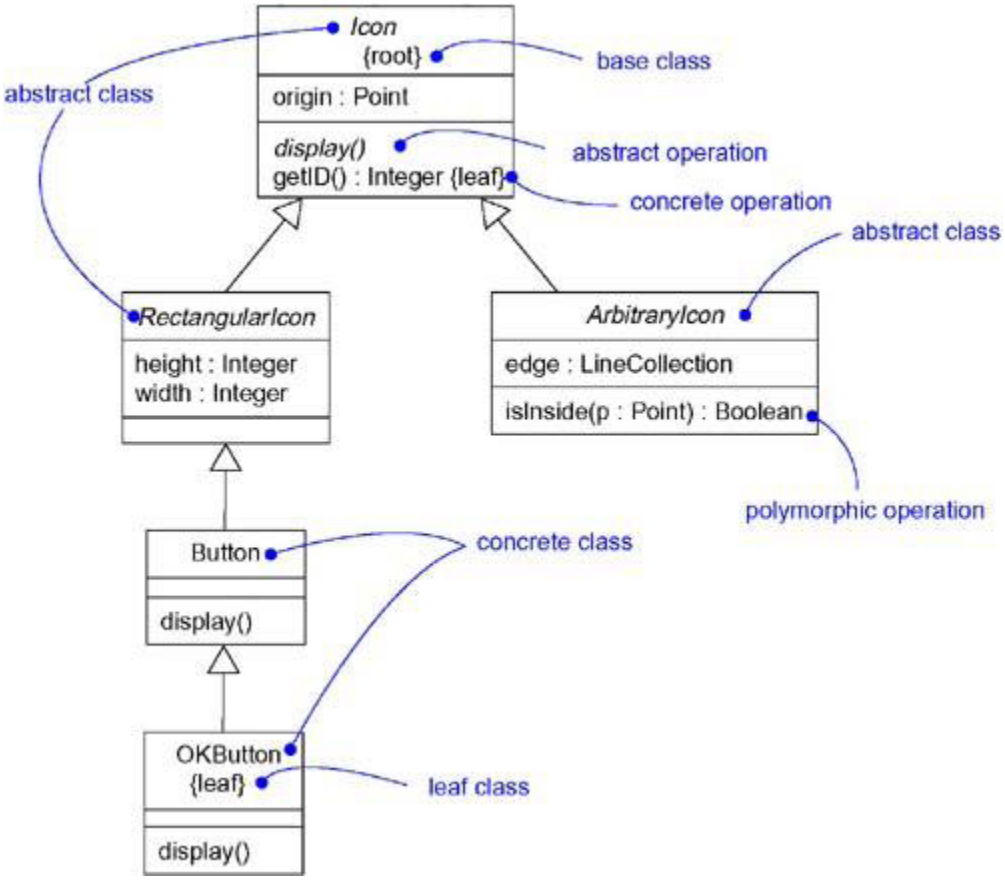

Figure:4 class Diagram indicating root,leaf,polymorphism and abstract

For example, *display* and *isInside* are both polymorphic operations. Furthermore, the operation Icon::display() is abstract, meaning that it is incomplete and requires a child to

supply an implementation of the operation. In the UML, you specify an abstract operation by writing its name in italics, just as you do for a class. By contrast, Icon::getID() is a leaf operation as indicated by the property leaf. This means that the operation is not polymorphic and may not be overridden.

Multiplicity

The number of instances a class may have is called its multiplicity. Multiplicity applies to attributes, as well. A class having single instance is called as a singleton class. It is indicated in Figure:5

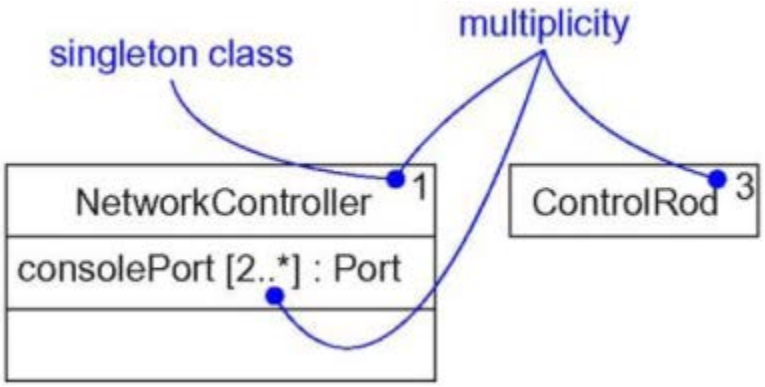

Figure:5 Multiplicity

Attributes

A class's structural features are indicated by its attributes.

The ***syntax of an attribute*** in UML is

[visibility] name [multiplicity] [: type] [= initial-value] [{property-string}]

Some legal attribute declarations are given in Table:3.

• origin	Name only
• + origin	Visibility and name
• origin : Point	Name and type
• head : *Item	Name and complex type
• name [0..1] : String	Name, multiplicity, and type
• origin : Point = (0,0)	Name, type, and initial value
• id : Integer {frozen}	Name and property

Table:3 Attributes

Three defined ***properties that can be used with attribute*** values are given in Table:4.

1. changeable	There are no restrictions on modifying the attribute's value.
2. addOnly	For attributes with a multiplicity greater than one, additional values may be added, but once created, a value may not be removed or altered.
3. frozen	The attribute's value may not be changed after the object is initialized.

Table:4 Attribute properties

Where default property is ‘changeable’.

Operations

A class’s behavioral features are indicated by its operations.

The UML ***distinguishes between operation and method***. An operation specifies a service that can be requested from any object of the class to affect behavior; a method is an implementation of an operation.

The ***syntax of an operation*** in the UML is

[visibility] name [(parameter-list)][: return-type] [{property-string}]

All legal operation declarations are indicated in Table:5

• display	Name only
• + display	Visibility and name
• set(n : Name, s : String)	Name and parameters
• getID() : Integer	Name and return type
• restart() {guarded}	Name and property

Table:5 Legal Operations

In an ***operation***‘s signature, you may provide zero or more parameters, each of which follows the***syntax.***

[direction] name : type [= default-value]

Direction of a parameter may be any of the following values given in Table:6.

• in	An input parameter; may not be modified
• out	An output parameter; may be modified to communicate information to the caller
• inout	An input parameter; may be modified

Table:6 Direction Parameter

In addition to the leaf property described earlier, there are four defined properties that can be used with operations. They are given in Table:7.

1. isQuery	Execution of the operation leaves the state of the system unchanged. In other words, the operation is a pure function that has no side effects.
2. sequential	Callers must coordinate outside the object so that only one flow is in the object at a time. In the presence of multiple flows of control, the semantics and integrity of the object cannot be guaranteed.
3. guarded	The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by sequentializing all calls to all of the object's guarded operations. In effect, exactly one operation at a time can be invoked on the object, reducing this to sequential semantics.
4. concurrent	The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by treating the operation as atomic. Multiple calls from concurrent flows of control may occur simultaneously to one object on any concurrent operation, and all may proceed concurrently with correct semantics; concurrent operations must be designed so that they perform correctly in the case of a concurrent sequential or guarded operation on the same object.

Table: 7 Operation_property

Template Classes

A template is a parameterized element. A template includes slots for classes,objects, and values, and these slots serve as the template's parameters. Every templates should be instantiated first. Instantiation involves binding these formal template parameters to actual ones. For a template class, the result is a concrete class that can be used just like any ordinary class.

The ***instantiation of a template class can be modelled in two ways***.

First done implicitly, by declaring a class whose name provides the binding. **Second**, explicitly by using a dependency stereotyped as bind, which specifies that the source instantiates the target template using the actual parameters. A template class is indicated in Figure: 6.

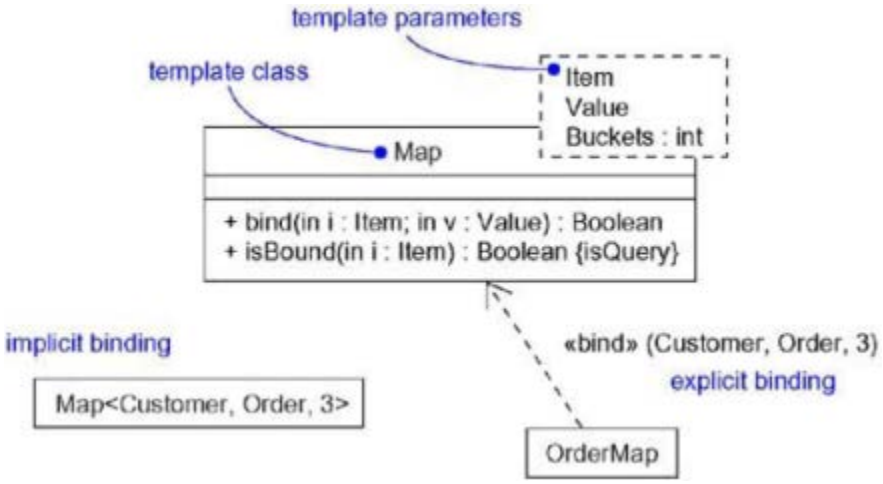

Figure:6 Template class

UML defines four standard stereotypes that apply to classes. They are indicated in Table:8.

1. metaclass	Specifies a classifier whose objects are all classes
2. powertype	Specifies a classifier whose objects are the children of a given parent
3. stereotype	Specifies that the classifier is a stereotype that may be applied to other elements
4. utility	Specifies a class whose attributes and operations are all class scoped

Table:8 Standard Prototypes

Source : http://praveenthomasln.wordpress.com/2012/02/25/advanced-classes-under-construction/