Department of Computer Science

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Tackling Design Patterns
Chapter 6: UML Class and Object Diagrams

Contents

6.1 Introduction

In object-oriented programming, classes do not function in isolation, they interact with each other. UML therefore needs to be able to model how classes interact and for structural UML diagrams, such as class and object diagrams, these interactions are modelled as relationships. Two broad categories of relationships exist, those which are used to model delegation and those used to model inheritance. Examples of delegation relationships are associations and dependencies, which have already been discussed. To model inheritance, the generalisation relationship has been defined in UML. Figure 1 provides an overview of the different relationships which can be modelled in UML [1].

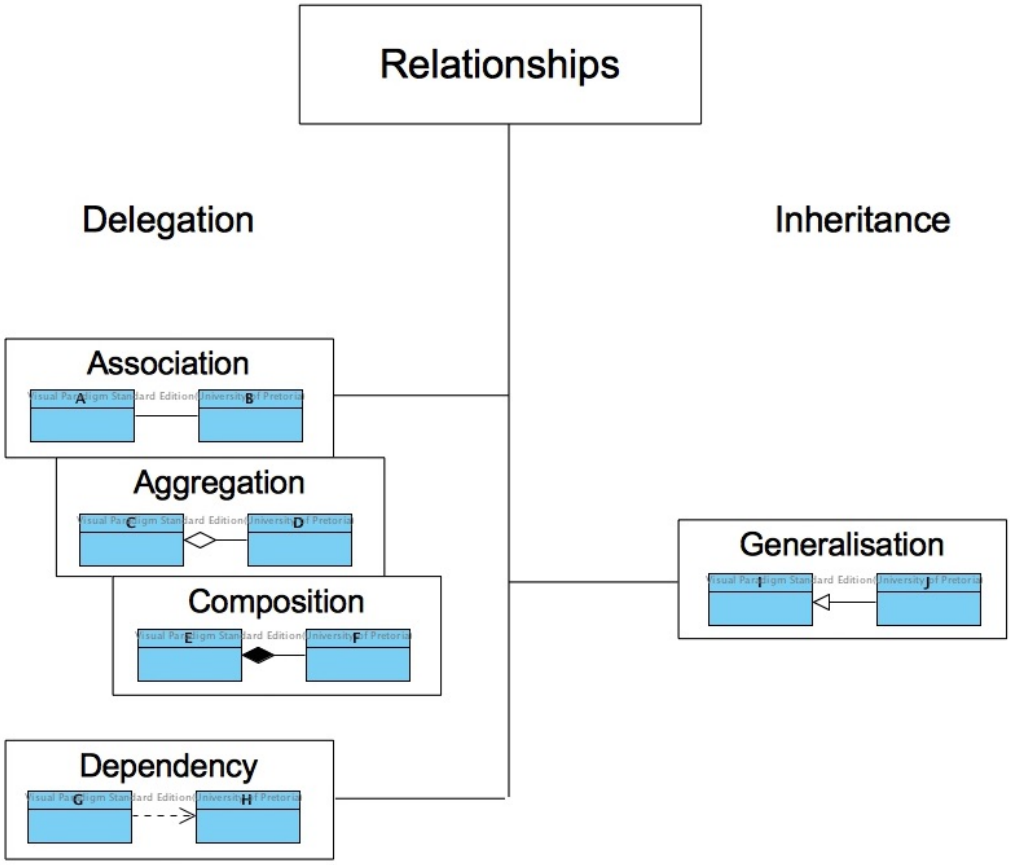

Figure 1: UML relationships

Modelling classes and delegation in UML was discussed in Chapter 2, specifically Sections 2.2.1 and 2.2.2 respectively. In this lecture note, modelling inheritance and objects will be the focus.

6.2 Modelling generalisation

Public inheritance has been used in Chapters 3, 4 and 5 in the code examples and presented in the corresponding UML Class diagrams for the examples. In Section 3.2.1 of Chapter

3, public inheritance was presented as an *is-a* relationship. A differentiation was also made between different types of operations in a class, namely: non-virtual, virtual and pure virtual. In Figure 2, `ConcreteClass` *is-a* `AbstractClass`. `ConcreteClass` inherits from `AbstractClass` and therefore there is a generalisation relationship between the two classes.

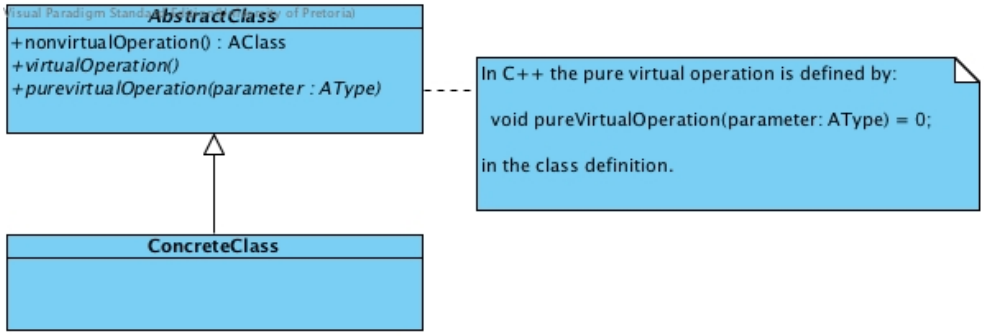

Figure 2: Modelling generalisation in UML

All operations in `AbstractClass` are public, this means that they can be called when an instance of `ConcreteClass`. `nonvirtualOperation`, as defined in `AbstractClass`, can be called by an object of `ConcreteClass`. The operation `virtualOperation` may be redefined in `ConcreteClass`, while `purevirtualOperation` must be implemented in `ClassOperation`. A class with a pure virtual operation cannot be instantiated and is therefore defined as abstract. Under the assumption that the non-virtual operation was not public, but protected or private, then a program or object calling operations of `ClassOperation` would not be access the operation. A operation within `ConcreteClass` would be able to call the operation if it were defined as protected.

6.3 Object diagrams

Object diagrams are derived from class diagrams and are therefore dependent on class diagrams. They represent an instance of a class or classes and are therefore seen as a static view of the class at a specific moment. Object diagrams are concrete in nature, they represent the real-word versus class diagrams which are abstract representations and are seen as a blue-print. An object diagram represents unlimited instances while class diagrams comprise of fixed classes. Object diagrams use the same basic relationships as class diagrams. Object diagrams can be used for forward and reverse engineering of code.

6.3.1 Modelling objects

Refer back to Figure 1 in Chapter 1 showing that both class and object diagrams are classified as UML structural diagrams depicting the relationships between classes and objects respectively.

An example of a diagram showing different styles a single object can be depicted in UML as given in Figure 3. Note the class which these objects are instances of, given in Figure 4, consists of an attribute with object scope (`attribute1`) and an attribute with class scope

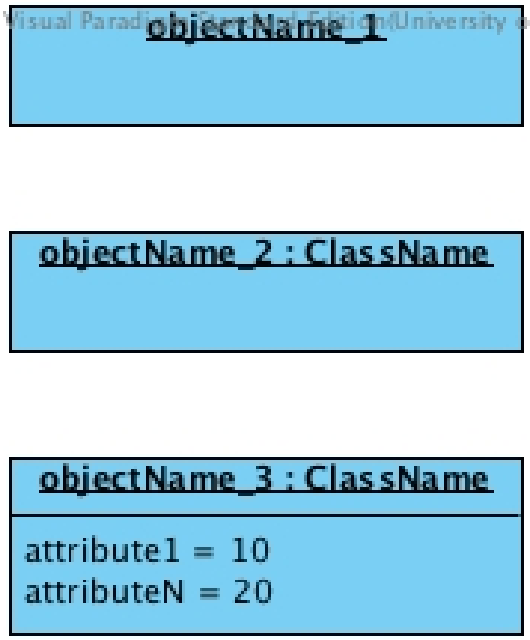

Figure 3: UML object diagrams

(`attributeN`) which will be defined as static. The scope distinction is not evident in the object diagrams.

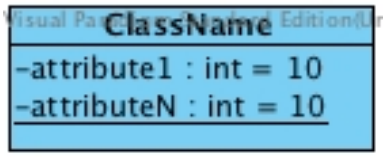

Figure 4: UML class diagram

To determine whether a diagram is a class or an object diagram, it is necessary to consider the name section of the diagram. If the name section is underlined (as with `objectName_1`), an object is being represented. The detail at which the object is being represented can also vary. `objectName_2` specifies which class this representation in as object of, while `objectName_3` details the values of the attributes of the object at specific moment. Operations do not feature in object diagrams.

6.3.2 Modelling relationships

Generalisation relationships are not modelled in object diagrams, but the modelling of associations and dependencies, or just showing that there is a relationship between objects is common *Need to complete*s

The are multiple ways of showing multiplicities in object diagrams. Consider the class diagram given in Figure 5. The class `University` has an attribute `students` which has been defined as an array of objects of class `Student`. From the multiplicity it is clear that

an object representing the class `University` may have 0 or more objects of class `Student`. Objects of class `Student` have an `age` and a `degree` attribute.

Figure 5: UML class diagram showing multiplicity

If the detail of each instance in the array is important, then each object of class `Student` associated with the object of class `University` should be shown in the object diagram. This is only practical when the number of objects in the association is not so high that the diagram becomes impossible to understand or when a selection of objects are to be depicted. Assuming we wish to show two students, Alice and Bob, and their affiliation to the the University of Funny Walks (UFW) - apologies to Monty Python [2]. The object diagram representing this situation is given in Figure 6.

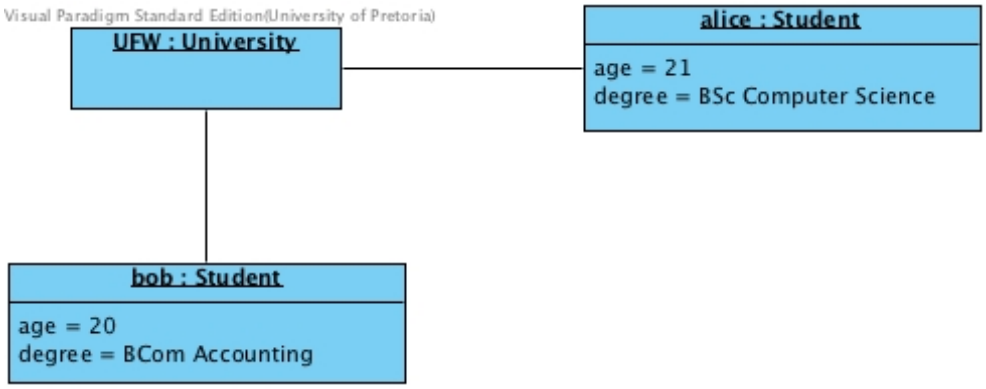

Figure 6: UML object diagram showing multiplicity with object detail

If only knowing which objects have been created is of relavance, then the object diagram can be shown as given in Figure 7 using an attribute textual notation.

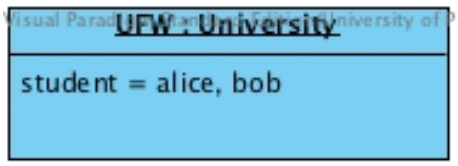

Figure 7: UML object diagram showing object attributes textual notation

Another notation in which the details are not important, yet the multiplicity is shown is given in Figure 8. Structures as these are not native to many modelling tools and therefore are not normally drawn as such or need to be manually drawn.

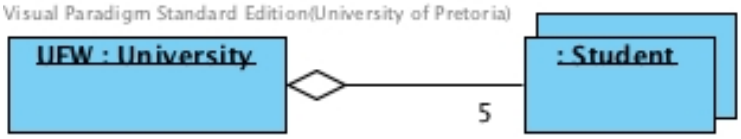

Figure 8: UML object diagram showing object attributes textual notation

6.3.3 Example

Consider the class diagram given in Figure 9 which is an implementation of the Template Method design pattern. Assume that the main program for the classes defined is given by:

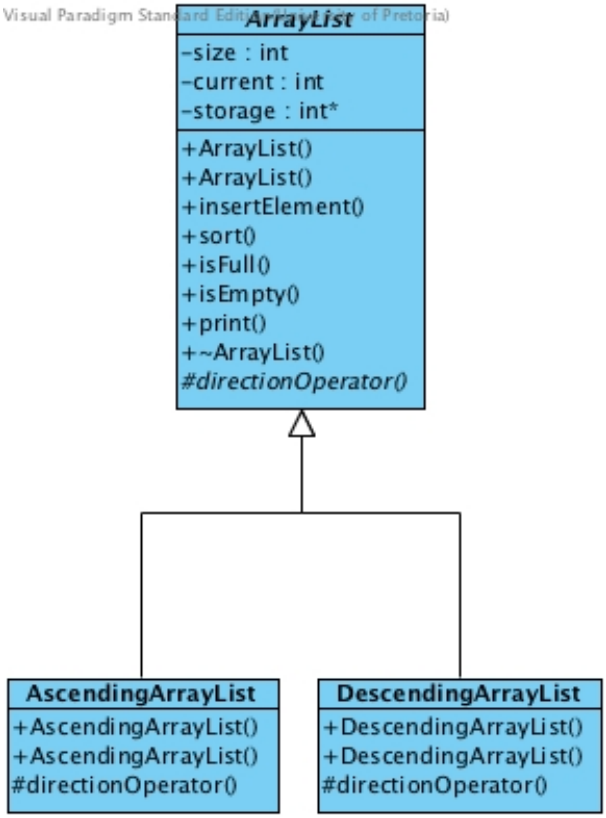

Figure 9: ArrayList, an implementation of the Template Method design pattern

```
1  #include <iostream>
2  #include "ArrayList.h"
3
4  using namespace std;
5
6  int main(){
7
8          ArrayList* arr = new DescendingArrayList(10);
9
10         arr->insertElement(10);
```

```
11          arr->insertElement(20);
12          arr->insertElement(15);
13          arr->insertElement(25);
14          arr->insertElement(5);
15
16          arr->print();
17          arr->sort();
18          arr->print();
19
20          delete arr;
21
22          return 0;
23  }
```

After the object has been created and instantiated in Line 8 of the main program, an object diagram for this state of the program can be drawn. This state is given in Figure 10. After 3 elements have been inserted into storage, that is at Line 12 in the code, the corresponding object diagram for the state of the program at that moment is given in Figure 11. After the elements and `arr` has been sorted, Line 17, the object at that particular moment is given in Figure 12. Note the change in the value of `current` in Figures 10 to 12.

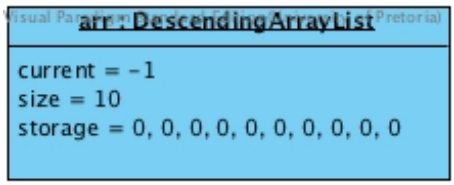

Figure 10: Object diagram on `arr` creation

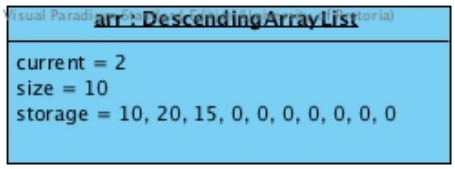

Figure 11: Object diagram after 3 elements have been inserted

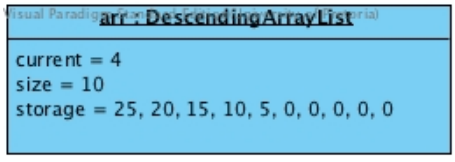

Figure 12: Object diagram after sorting

6.3.4 Object diagrams when modelling behaviour

Objects play a role when behaviour is modelled, using UML Sequence and Communication diagrams. What is depicted in these diagrams is not the state of the object at a given moment, but the communication between objects.

References

[1] Object Management Group. Unified Modeling Language (OMG UML) Version 2.5, 2015. URL `http://www.omg.org/spec/UML/2.5/PDF/`. [Online; accessed 1 August 2015].

[2] Monty Python. Monty Python's - The Ministry of Silly Walks, 2015. URL `http://www.thesillywalk.com`. [Online; accessed 1 August 2015].