

Classes



In this chapter you will be introduced to **Classes**.
Classes group related objects together.

What are Classes?

Instead of specifying similar objects over and over again, **classes** provide a template for objects. Classes are used to group related properties. Think of a class as a detailed description of an object.

Whilst a class is very similar to an object, it is however not an object, but it contains all the information needed to create an object. It is like the relationship between a blueprint and a house or a word processing template and a document.

Let us look at our Sloopy family again in Figure 1 – Eddy, Nina and Dima. The class, which we have named *Sloopy*, contains three objects (Eddy, Nina and Dima), objects as defined in the last section. There is no limit of the number of objects that can belong to a class unless when we are modelling an application which has constraints (for example, a school class might have a limit of 32 pupils or a soccer team has a limit of 11 players).



Figure 1: The *Sloopy* class with three objects

In order to represent / model the three Sloopies, a *Sloopy* class is created, which contains properties and types. **Types** can have certain different forms of value. Typical types are text, number or date, but they can be anything user-defined such as colour. Therefore our *Sloopy* class can look as follows:

Property	Type
Hair	Colour
eyes	Colour
age	Number
jumper	Colour
shoes	Colour
belt	True/False

Figure 2: The *Sloopy* class with its properties and type

A type is a classification of data. It indicates a set of values that have the same general meaning or intended purpose. For example, hair and eyes are described by a colour, age is described by a number and each object (Eddy, Nina or Dima) from the *Sloopy* class either wears a belt or not.

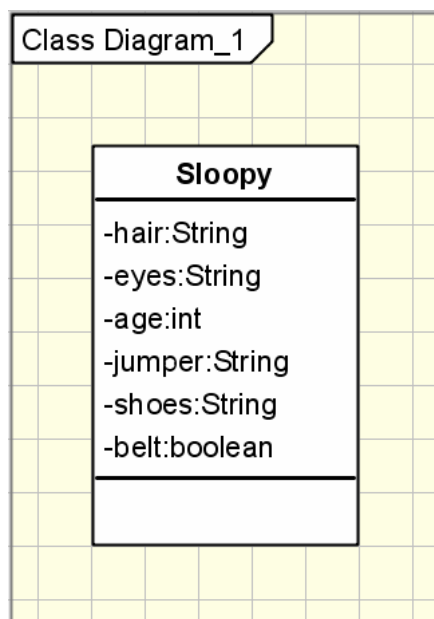


Figure 3: A Class Diagram drawn using Poseidon

As mentioned before, the role of the Unified Modelling Language (UML) is to represent everything that is being modelled when developing applications (see Overview Chapter). To represent classes, so called **Class Diagrams** have to be used. Class diagrams can be drawn by hand or by using software programs.

Poseidon is used throughout this course to create all UML diagrams. Class diagrams are represented as rectangular boxes, which consists of three parts (see Figure 3).

The top part shows the *class name* (*Sloopy*), the middle part represents the properties/attributes and the bottom part shows behaviour, which will be introduced at a later stage and can be left empty for now.

UML supports a wide range of types, which are mainly taken from the **Java programming languages**. For the purpose of this lesson,

1. Colour is represented by *String* (which is text),
2. Number by *int* (which is an integer)
3. Yes/No by *boolean* (which is either true/false).

We can also represent this class in what is called **pseudo code** (an outline of a program, written in a form that can easily be converted into real programming statements). The pseudo code shows a textual representation of the graphical UML class diagram. The *Sloopy* class could be written as follows using pseudo code:

```
CLASS Sloopy
ATTRIBUTES:
    hair: String
    eyes: String
    age: int
    jumper: String
    shoes: String
    belt: boolean
```

Summary

At this stage, classes are made up of properties. These classes can be represented or modelled in many ways.

1. They can be described using natural language
2. They can be visualised using UML, either by hand or produced by software that supports UML.
3. They can be represented by pseudo code.
4. They can be represented by an Object-Oriented Programming Language.



In the next chapter we will look at instances, the means of using objects and classes